Block

```
int n, m;
int vis[25][25], maze[25][25];

int used(int x, int y) { return (x > 0 && x <= n && y > 0 && y <= m) ; }

struct PT {
    int x, y;
    PT () {}
    PT (int x, int y) : x(x) , y(y) {}
    bool operator < (const PT &o) const { return (x == o.x) ? (y < o.y) :  (x < o.x) ; }
}   ;

struct BLOCK {
    vector <PT> p;
    int mx, my;
    bool operator == (const BLOCK &o) const {
        for (int i = 0; i < p.size(); i++) if (p[i].x != o.p[i].x || p[i].y != o.p[i].y) return 0;
        return 1;
    }
    void bfs(int sx, int sy) {
        queue <PT> q;
        int X[4] = {0, 1, 0,-1};
        int Y[4] = {1, 0,-1, 0};

        for (q.push(PT(sx, sy)), vis[sx][sy] = 1; q.empty() == 0; ) {
            int x = q.front().x, y = q.front().y; q.pop();
            p.push_back(PT(x, y));
            for (int t = 0; t < 4; t++) {
                int i = x + X[t], j = y + Y[t];
                if (vis[i][j] == 0 && used(i, j) && maze[i][j] == maze[sx][sy])
                    vis[i][j] = 1, q.push(PT(i, j));
            }
        }
    }
    void adjust() {
        int x = 99, y = 99;
        for (int i = 0; i < p.size(); i++) x = min(x, p[i].x), y = min(y, p[i].y);
        for (int i = 0; i < p.size(); i++) p[i].x -= x, p[i].y -= y;
        mx = my = 0;
        for (int i = 0; i < p.size(); i++) mx = max(mx, p[i].x), my = max(my, p[i].y);
        sort(p.begin(), p.end());
    }
    void full(int x, int y, char c) {
        for (int i = 0; i < p.size(); i++) maze[p[i].x + x][p[i].y + y] = c;
    }
    BLOCK rotate() {
        BLOCK ret;
        for (int i = 0; i < p.size(); i++) ret.p.push_back(PT(p[i].y, mx - p[i].x));
        return ret;
    }
    BLOCK reflex() {
        BLOCK ret;
        for (int i = 0; i < p.size(); i++) ret.p.push_back(PT(mx - p[i].x, p[i].y));
        return ret;
    }
}   ;
```

Cube

```
struct FACE {
    int a[3][3];
    bool operator == (const FACE &o) {
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                if (a[i][j] != o.a[i][j])
                    return 0;
        return 1;
    }
    void read(int c) {
        for (int i = 0, id = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                a[i][j] = c * 9 + id, ++id;
    }
}   ;

struct CUBE {
    FACE s[6];
    CUBE () {
        for (int i = 0; i < 6; i++) s[i].read(i);
    }
    bool operator == (const CUBE &o) {
        for (int i = 0; i < 6; i++) if ((s[i] == o.s[i]) == 0) return 0;
        return 1;
    }
    void rotate(int ty) {
        int X[6][20]={
                { 0, 1, 2, 9,10,11,18,19,20,27,28,29,42,43,44,41,38,37,36,39},
                { 6, 7, 8,15,16,17,24,25,26,33,34,35,45,46,47,50,53,52,51,48},
                {36,39,42, 0, 3, 6,45,48,51,26,23,20,27,28,29,32,35,34,33,30},
                {44,41,38,18,21,24,53,50,47, 8, 5, 2, 9,10,11,14,17,16,15,12},
                {42,43,44, 9,12,15,47,46,45,35,32,29, 0, 1, 2, 5, 8, 7, 6, 3},
                {38,37,36,27,30,33,51,52,53,17,14,11,18,19,20,23,26,25,24,21}};
        int Y[6][20]={
                { 9,10,11,18,19,20,27,28,29, 0, 1, 2,44,41,38,37,36,39,42,43},
                {33,34,35, 6, 7, 8,15,16,17,24,25,26,51,48,45,46,47,50,53,52},
                {26,23,20,36,39,42, 0, 3, 6,45,48,51,33,30,27,28,29,32,35,34},
                { 8, 5, 2,44,41,38,18,21,24,53,50,47,15,12, 9,10,11,14,17,16},
                {35,32,29,42,43,44, 9,12,15,47,46,45, 6, 3, 0, 1, 2, 5, 8, 7},
                {17,14,11,38,37,36,27,30,33,51,52,53,24,21,18,19,20,23,26,25}};

        CUBE tmp = *this;
        for (int i = 0, k = ty >> 1; i < 20; i++) {
            int x = X[k][i], y = Y[k][i];
            if (ty & 1) s[y / 9].a[y % 9 / 3][y % 3] = tmp.s[x / 9].a[x % 9 / 3][x % 3];
            else s[x / 9].a[x % 9 / 3][x % 3]=tmp.s[y / 9].a[y % 9 / 3][y % 3];
        }
    }
}   cube;

int X[9][20]={
                {42,43,44, 9,12,15,47,46,45,35,32,29, 0, 1, 2, 5, 8, 7, 6, 3},
                {48,49,50,16,13,10,41,40,39,28,31,34},
                {17,14,11,38,37,36,27,30,33,51,52,53,24,21,18,19,20,23,26,25},
                {26,23,20,36,39,42, 0, 3, 6,45,48,51,33,30,27,28,29,32,35,34},
                {46,49,52,25,22,19,37,40,43, 1, 4, 7},
                {44,41,38,18,21,24,53,50,47, 8, 5, 2, 9,10,11,14,17,16,15,12},
                { 0, 1, 2, 9,10,11,18,19,20,27,28,29,42,43,44,41,38,37,36,39},
                { 3, 4, 5,12,13,14,21,22,23,30,31,32},
                {33,34,35, 6, 7, 8,15,16,17,24,25,26,51,48,45,46,47,50,53,52} };
int Y[9][20]={
                {35,32,29,42,43,44, 9,12,15,47,46,45, 6, 3, 0, 1, 2, 5, 8, 7},
                {16,13,10,41,40,39,28,31,34,48,49,50},
                {38,37,36,27,30,33,51,52,53,17,14,11,18,19,20,23,26,25,24,21},
                {36,39,42, 0, 3, 6,45,48,51,26,23,20,27,28,29,32,35,34,33,30},
                {25,22,19,37,40,43, 1, 4, 7,46,49,52},
                { 8, 5, 2,44,41,38,18,21,24,53,50,47,15,12, 9,10,11,14,17,16},
                { 9,10,11,18,19,20,27,28,29, 0, 1, 2,44,41,38,37,36,39,42,43},
                {12,13,14,21,22,23,30,31,32, 3, 4, 5},
                { 6, 7, 8,15,16,17,24,25,26,33,34,35,45,46,47,50,53,52,51,48} };
```

Min_Rep

```
int MinRep(int *str, int len) {
    int i = 0, j = 1, k = 0;
```

```
    while (i < len && j < len && k < len)  {
        int l1 = i + k, l2 = j + k;
        if (l1 >= len) l1 -= len;
        if (l2 >= len) l2 -= len;

        int t = str[l1] - str[l2];
        if (t == 0) k++;
        else {
            if (t > 0) i = i + k + 1;
            else j = j + k + 1;
            if (i == j) j++;
            k = 0;
        }
    }
    return i < j ? i : j;
}
```

Manacher

```
//for i  len[i<<1] as odd palindrome
//      len[i<<1|1] as even palindrome
void palindrome(char cs[], int len[], int n) { //len[i] means the max palindrome length centered i/2
    for (int i = 0; i < n * 2; ++i) len[i] = 0;
    for (int i = 0, j = 0, k; i < n * 2; i += k, j = max(j - k, 0)) {
        while (i - j >= 0 && i + j + 1 < n * 2 && cs[(i - j) / 2] == cs[(i + j + 1) / 2]) j++;
        len[i] = j;
        for (k = 1; i - k >= 0 && j - k >= 0 && len[i - k] != j - k; k++)
            len[i + k] = min(len[i - k], j - k);
    }
}
```

DA

```
#define maxn 1000001
int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
int cmp(int *r,int a,int b,int l)
{return r[a]==r[b]&&r[a+l]==r[b+l];}
void da(int *r,int *sa,int n,int m)
{
     int i,j,p,*x=wa,*y=wb,*t;
     for(i=0;i<m;i++) ws[i]=0;
     for(i=0;i<n;i++) ws[x[i]=r[i]]++;
     for(i=1;i<m;i++) ws[i]+=ws[i-1];
     for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i;
     for(j=1,p=1;p<n;j*=2,m=p)
     {
       for(p=0,i=n-j;i<n;i++) y[p++]=i;
       for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
       for(i=0;i<n;i++) wv[i]=x[y[i]];
       for(i=0;i<m;i++) ws[i]=0;
       for(i=0;i<n;i++) ws[wv[i]]++;
       for(i=1;i<m;i++) ws[i]+=ws[i-1];
       for(i=n-1;i>=0;i--) sa[--ws[wv[i]]]=y[i];
       for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
       x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
     }
     return;
}
int rank[maxn],height[maxn];
void calheight(int *r,int *sa,int n)
{
     int i,j,k=0;
     for(i=1;i<=n;i++) rank[sa[i]]=i;
     for(i=0;i<n;height[rank[i++]]=k)
     for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);
     return;
}
int RMQ[maxn];
int mm[maxn];
int best[20][maxn];
void initRMQ(int n)
{
     int i,j,a,b;
     for(mm[0]=-1,i=1;i<=n;i++)
     mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
     for(i=1;i<=n;i++) best[0][i]=i;
     for(i=1;i<=mm[n];i++)
     for(j=1;j<=n+1-(1<<i);j++)
     {
       a=best[i-1][j];
       b=best[i-1][j+(1<<(i-1))];
       if(RMQ[a]<RMQ[b]) best[i][j]=a;
       else best[i][j]=b;
     }
     return;
}
int askRMQ(int a,int b)
{
    int t;
    t=mm[b-a+1];b-=(1<<t)-1;
    a=best[t][a];b=best[t][b];
    return RMQ[a]<RMQ[b]?a:b;
}
int lcp(int a,int b)
{
    int t;
    a=rank[a];b=rank[b];
    if(a>b) {t=a;a=b;b=t;}
    return(height[askRMQ(a+1,b)]);
}


/*
对于一个从 0->n-1 的字符串
SA[i]表示排在第 i 位的后缀起始地址为 SA[i]   i 从 1 -> n (SA[0]为放在最后一位的 0)
Rank 从 0->n-1
Height[i]表示第 i 名的后缀和第 i-1 名的后缀的最长公共前缀  且 height[1] = 0

分组:
//height[n+1]=-1;
for (i = 2; i <= n; i=j+1){
    while(i<=n&&height[i] < u)i++;
    for(j=i;height[j]>=u;j++);
    for(k=i-1;k<j;k++){
    }
}

使用 da 或 dc3 前生成一个 r 数组, r 的[0,n-1]对应字符串的[0,n-1],
r[n]=0
调用 da(r,sa,n+1,m)
调用 calheight(r,sa,n)得到 height 数组, 建议在后面加上一句:height[n+1]=-1
m 一般取 128.如果碰到字符串不止大小写字母, 一般用 328,r[0,n-1]=str[0,n-1]+200
调用 lcp 前, 需要把 height 数组移给 RMQ 数组, 调用 initRMQ(n);

如果需要用 iostream, 把 ws 改成_ws
*/
```

ExtendKMP

```
// ex[i]为满足 A[i..i+z-1]==B[0..z-1]的最大的 z 值
    lenA = strlen(A); lenB = strlen(B);
    next[0] = lenB; next[1] = lenB - 1;
    re(i, lenB-1) if (B[i] != B[i + 1]) {next[1] = i; break;}
    int j, k = 1, p, L;
    re2(i, 2, lenB) {
        p = k + next[k] - 1; L = next[i - k];
```

```
        if (i + L <= p) next[i] = L; else {
            j = p - i + 1;
            if (j < 0) j = 0;
            while (i + j < lenB && B[i + j] == B[j]) j++;
            next[i] = j; k = i;
        }
    }
    int minlen = lenA <= lenB ? lenA : lenB; ex[0] = minlen;
    re(i, minlen) if (A[i] != B[i]) {ex[0] = i; break;}
    k = 0;
    re2(i, 1, lenA) {
        p = k + ex[k] - 1; L = next[i - k];
        if (i + L <= p) ex[i] = L; else {
            j = p - i + 1;
            if (j < 0) j = 0;
            while (i + j < lenA && j < lenB && A[i + j] == B[j]) j++;
            ex[i] = j; k = i;
        }
    }
```

SAM

```
struct sanode{
    sanode *f, *ch[26];
    int ml;
} *root, *tail, pool[MAXN], *q[MAXN];

int tot;
char s[MAXN>>1];

void init(){
    memset(pool,0,sizeof(pool));
    tot = 0;
    root = tail = &pool[tot++];
}

void add(int c){
    sanode *p = tail, *np = &pool[tot++];
    np->ml = tail->ml + 1;
    for(;p&&!p->ch[c];p=p->f) p->ch[c] = np;
    tail = np;
    if(!p) np->f = root;
    else
        if(p->ch[c]->ml == p->ml+1) np->f = p->ch[c];
        else{
            sanode *q = p->ch[c], *r = &pool[tot++];
            *r = *q;
            r->ml = p->ml + 1;
            q->f = np->f = r;
            for(;p&&p->ch[c]==q;p=p->f) p->ch[c] = r;
        }
}
```

AC 自动机

```
    //得到fail 指针
    void bfs(){
        int head, tail, v, i, u;
        head = 0;
        tail = -1;
        for (i = 0; i < CHILD_NUM; i++){
            v = chd[0][i];
            if (!v) continue;
            q[++tail] = v;
            fail[v] = 0;
        }
        while(head <= tail){
            u = q[head];
            for (i = 0; i < CHILD_NUM; i++){
                if (chd[u][i]){
                    v = chd[u][i];
                    q[++tail] = v;
                    fail[v] = chd[fail[u]][i];
                    //以下一行代码要根据题目所给 val 的含义来写
                    //val[v] |= val[fail[v]];
                } else
                chd[u][i] = chd[fail[u]][i];
            }
            ++head;
        }
    }

```

KM

```
const int NN = 405;
const int inf = 0x0f0f0f0f;

int sx[NN], sy[NN];
int mth[NN], w[NN][NN], lx[NN], ly[NN], n, m; //n：左集元素个数； m：右集元素个数
void init() { memset(w, 0, sizeof(w)); } //不一定要，求最小值一般要初始化为负无穷!
int dfs(int u) {
    sx[u] = 1;
    for (int v = 1; v <= m; v++)
        if (!sy[v] && lx[u] + ly[v] == w[u][v])
            if (sy[v] = 1, mth[v] == 0 || dfs(mth[v]))
                return mth[v] = u, 1;
    return 0;
}
int KM() {
    int i, j, k, sum=0;
    memset(ly, 0, sizeof(ly));
    memset(mth, 0, sizeof(mth));
    for (int i = 1; i <= n; i++) {
        lx[i] = -inf;
        for (int j = 1; j <= m; j++)
            if (lx[i] < w[i][j]) lx[i] = w[i][j];
    }
    for (int i = 1; i <= n; i++)
        while (1) {
            memset(sx, 0 sizeof(sx));
            memset(sy, 0, sizeof(sy));
            if (dfs(i)) break;
            int d = inf;
            for (int j = 1; j <= n; j++)
                for (int k = 1; sx[j] && k <= m; k++)
                    if (!sy[k])
                        d = min(d, lx[j] + ly[k] - w[j][k]);
            if (d == inf) return -1;
            for (int j = 1; j <= n; j++) if (sx[j]) lx[j] -= d;
            for (int j = 1; j <= m; j++) if (sy[j]) ly[j] += d;
        }
    for (int i = 1; i <= m; i++)
        if (mth[i])
            sum += w[mth[i]][i];
    return sum;
}
```

Dinic

```
const int N = 100000;
const int M = 100000;
namespace MaxFlow{
    int sz;
    int gcnt, ghead[N], to[M], cap[M], nx[M], cur[N];
    int source, target, flow, pre[N], sign;
```

```
022
023     void addEdge(int u, int v, int w){
024         nx[gcnt] = ghead[u];
025         to[gcnt] = v;
026         cap[gcnt] = w;
027         ghead[u] = gcnt++;
028     }
029
030     void insert(int u, int v, int w){
031         addEdge(u, v, w);
032         addEdge(v, u, 0);
033     }
034
035     int level[N], que[N];
036     bool bfs(int s, int t){
037         memset(level, -1, sizeof(level));
038         sign = t;
039         level[t] = 0;
040         int tail = 0, head = 0;
041         que[tail++] = t;
042         while(head!=tail && level[s]==-1){
043             int v = que[head++];
044             for(int iter = ghead[v];iter!=-1;iter=nx[iter]){
045                 if(cap[iter^1]>0&&level[to[iter]]==-1){
046                     level[to[iter]] = level[v]+1;
047                     que[tail++] = to[iter];
048                 }
049             }
050         }
051         return level[s]!=-1;
052     }
053
054     inline void push(){
055         int delta = INT_MAX, u, p;
056         for(u=target;u!=source;u=to[p]){
057             p = pre[u];
058             delta = min(delta, cap[p]);
059             p^=1;
060         }
061         for(u=target;u!=source;u=to[p]){
062             p = pre[u];
063             cap[p] -= delta;
064             if(!cap[p]) sign = to[p^1];
065             cap[p^=1]+=delta;
066         }
067         flow += delta;
068     }
069
070     void dfs(int u){
071         if(u==target){
072             push();
073             return ;
074         }
075         for(int &iter=cur[u];iter!=-1;iter=nx[iter]){
076             if(cap[iter]>0 && level[u]==level[to[iter]]+1){
077                 pre[to[iter]] = iter;
078                 dfs(to[iter]);
079                 if(level[sign]>level[u]) return;
080                 sign = target;
081             }
082         }
083         level[u] = -1;
084     }
085
086     void initNetwork(int nodes){
087         sz = nodes;
088         gcnt = 0;
089         memset(ghead,-1,sizeof(ghead));
090     }
091
092     int maxFlow (int s, int t){
093         source = s;
094         target = t;
095         flow =0;
096         while (bfs(source, target)) {
097             for (int i = 0; i < sz; ++ i) {
098                     cur [i] = ghead[i];
099             }
100             dfs(source);
101         }
102         return flow ;
103     }
104 }
105 using namespace MaxFlow;
106 //!!! maxFlow end
107
108 void init(){
109     //initNetwork ...
110     // set s&&t
111     // maxFlow(s,t)
112 }
113
114 int main(){
115     init();
116     return 0;
117 }
```

ISAP

```
01 #include <iostream>
02 #include <cstring>
03 #include <cstdio>
04
05 using namespace std;
06
07 const int NN = 400005;
08 const int inf = 0x0f0f0f0f;
09 struct EDGE {
10     int i, c;
11     EDGE *next, *ani;
12     EDGE () {}
13     EDGE (int i, int c, EDGE *next, EDGE *ani) : i(i), c(c), next(next), ani(ani) {}
14 }    *Edge[NN], e[NN << 1];
15
16 void _addedge(int i, int j, int c, EDGE &e1, EDGE &e2)
17 {
18     e1 = EDGE(j, c, Edge[i], &e2); Edge[i] = &e1;
19     e2 = EDGE(i, 0, Edge[j], &e1); Edge[j] = &e2;
20 }
21
22 int Dfn[NN], Gap[NN], S, E, CNT;
23 int ISAP(int n, int flow)
24 {
25     if (n == E) return flow;
26     int i, tab = CNT, vary, now = 0;
27     for (EDGE *p = Edge[n]; p; p = p->next)
28         if (p->c) {
29             if (Dfn[n] == Dfn[i = p->i] + 1)
30                 vary = ISAP(i, min(p->c, flow - now)),
31                 p->c -= vary, p->ani->c += vary, now += vary;
32             if (Dfn[S] == CNT) return now;
33             if (p->c) tab = min(tab, Dfn[i]);
34             if (now == flow) break;
35         }
36     if (now == 0) {
```

```
        if (--Gap[Dfn[n]] == 0) Dfn[S] = CNT;
        Gap[Dfn[n] = tab + 1]++;
    }
    return now;
}
int Maxflow(int s,int end)
{
    int flow = 0;
    //S=s, E=end, CNT=E+1;
    //memset(Edge, 0, sizeof(Edge));
    memset(Gap, 0, sizeof(Gap));
    memset(Dfn, 0, sizeof(Dfn));
    for (Gap[0] = CNT; Dfn[S] < CNT; ) flow += ISAP(S, inf);
    return flow;
}
```

MCMF

```
const int NN = 5005;
const int inf = 0x0f0f0f0f;

struct EDGE {
    int i, c, d;
    EDGE *next, *ani;
    EDGE() {}
    EDGE(int i, int c, int d, EDGE *next, EDGE *ani) : i(i), c(c), d(d), next(next), ani(ani) {}
}    *Edge[NN], *Path[NN], e[NN << 3];

void _addedge(int i, int j, int c, int d, EDGE &e1, EDGE &e2) {
    e1 = EDGE(j, c, d, Edge[i], &e2); Edge[i] = &e1;
    e2 = EDGE(i, 0, -d, Edge[j], &e1); Edge[j] = &e2;
}

int ds[NN], inq[NN];
int SPFA(int s, int end) {
    int i, j; queue <int> q;
    memset(ds, 0x0f, sizeof(ds));
    memset(inq, 0, sizeof(inq));
    for (ds[s] = 0, inq[s] = 1, q.push(s); !q.empty(); ) {
        i = q.front(), q.pop(), inq[i] = 0;
        for (EDGE *p = Edge[i]; p; p = p->next)
            if (p->c && ds[j = p->i] > ds[i] + p->d)
                if (Path[j] = p, ds[j] = ds[i] + p->d, inq[j] == 0)
                    inq[j] = 1, q.push(j);
    }
    return ds[end] < inf;
}
int Min_cost_flow(int s, int end) {
    int i, cost = 0, flow;
    while (SPFA(s, end)) {
        for (flow = inf, i = end; i != s; i = Path[i]->ani->i) flow = min(flow, Path[i]->c);
        for (cost + =ds[end] * flow, i = end; i != s; i = Path[i]->ani->i) Path[i]->c -= flow, Path[i]
->ani->c += flow;
    }
    return cost;
}
```

EBC

```
struct EBC {
    static const int M = 255 ;
    int con[M][M], mth[M], fa[M], base[M], path[M], inb[M], inq[M];
    int n;

    void init() {
        memset(con, 0, sizeof(con));
        memset(mth, 0, sizeof(mth));
    }
    void _addedge(int i, int j) { con[i][j] = con[j][i] = 1; }

    int lca(int i, int j) {
        memset(path, 0, sizeof(path));
        for ( ; i; i = fa[mth[i]]) i = base[i], path[i] = 1;
        for ( ; j; j = fa[mth[j]]) if (j = base[j], path[j]) return j;
    }
    void reset(int i, int anc) {
        for (int j; i != anc; i = j) {
            j = mth[i];
            inb[base[j]] = inb[base[i]] = 1;
            if (j = fa[j], base[j] != anc) fa[j] = mth[i];
        }
    }
    void contract(int x, int y, queue <int> &q) {
        memset(inb, 0, sizeof(inb));
        int anc = lca(x, y);
        reset(x, anc), reset(y, anc);
        if (base[x] != anc) fa[x] = y;
        if (base[y] != anc) fa[y] = x;
        for (int i = 1; i <= n; i++)
            if (inb[base[i]]) {
                if (base[i] = anc, inq[i] == 0) q.push(i), inq[i] = 1;
            }
    }
    int dfs(int s) {
        queue <int> q;
        memset(fa, 0, sizeof(fa));
        memset(inq, 0, sizeof(inq));
        for (int i = 1; i <= n; i++) base[i] = i;

        for (q.push(s), inq[s] = 1; q.empty() == 0; ) {
            int i = q.front(); q.pop();
            for (int j = 1; j <= n; j++)
                if (con[i][j] && base[j] != base[i] && mth[i] != j) {
                    if (j == s || (mth[j] && fa[mth[j]])) contract(i, j, q);
                    else if (fa[j] == 0) {
                        if (fa[j] = i, mth[j]) q.push(mth[j]), inq[mth[j]] = 1;
                        else return augment(j), 1;
                    }
                }
        }
        return 0;
    }
    void augment(int i) {
        for (int j, t; i; i = t) j = fa[i], t = mth[j], mth[i] = j, mth[j] = i;
    }
    int solve() {
        int ans = 0;
        for (int i = 1; i <= n; i++) ans += (mth[i] == 0 && dfs(i));
        return ans;
    }
}    ebc;
```

Stoer_Wagner

```
const int NN = 505;
const int inf = 0x0f0f0f0f;

int cap[NN][NN], dfn[NN], wage[NN];
void _addedge(int i, int j, int c) { cap[i][j] += c; cap[j][i] += c; }

int Stoer_Wagner(int n) {
    int mincut = inf;
    for (int i = 1; i <= n; i++) dfn[i] = i;
    for ( ; n > 1; n--) {
        memset(wage, 0, sizeof(wage));
        for (int i = 1; i <= n; i++, swap(dfn[i], dfn[now])) {
```

```
            int now = i + 1, t;
            for (int j = i + 1; j <= n; j++)
                if (t = dfn[j], wage[t] += cap[dfn[i]][t], wage[dfn[now]] < wage[t])
                    now = j;
        }
        mincut = min(mincut, wage[dfn[n]]);
        for (int i = 1; i <= n - 1; i++) {
            int now = cap[dfn[i]][dfn[n]];
            cap[dfn[i]][dfn[n - 1]] += now;
            cap[dfn[n - 1]][dfn[i]] += now;
        }
    }
    return mincut;
}
```

Tarjan

```
/**hint
    templete: tarjan_point for directed graph
    对于割点，templete 1 的处理方式是将其属于其中的某个连通分量，
    如果需要对每个连通分量都需要操作一次，见 templete 2
*/
#define N
#define M

int to[M], nx[M], ghead[N];
int gcnt;
void addedge(int l, int r){
    to[gcnt] = r;
    nx[gcnt] = ghead[l];
    ghead[l] = gcnt++;
}

int dfn[N], low[N], sta[N], Blo[N];
bool instack[N];
int block, mark, now;

void tarjan(int u){
    int iter, v;
    dfn[u] = low[u] = ++mark;
    instack[sta[++now]=u] = 1;
    for(iter=ghead[u];iter!=-1;iter=nx[iter]){
        if(!dfn[v=to[iter]]){
            tarjan(v), low[u] = min(low[u],low[v]);
        } else
        if(instack[v]) low[u] = min(low[u], dfn[v]);
    }
    if(low[u]==dfn[u] && ++block)
        do Blo[v=sta[now--]]=block,instack[v]=0;
        while(v!=u);
}

void solve(int n){
    int i;
    block = 0, mark = 0, now = 0;
    memset(instack,0,sizeof(instack));
    memset(dfn,0,sizeof(dfn));
    memset(Blo,0,sizeof(Blo));
    for(i=1;i<=n;i++)
    if(!dfn[i])tarjan(i);

}

/**
    templete 2
*/
#include <cstdio>
#include <cstring>
#include <iostream>
#define N
#define M
using namespace std;

int to[M], nx[M], ghead[N];
int gcnt;
void addedge(int l, int r){
    to[gcnt] = r;
    nx[gcnt] = ghead[l];
    ghead[l] = gcnt++;
}

int dfn[N], low[N], sta[N], Blo[N], tmp[N];
bool instack[N];
int block, mark, now;

void tarjan(int u){
    int iter, v;
    dfn[u] = low[u] = ++mark;
    instack[sta[++now]=u] = vis[u] = 1;
    for(iter=ghead[u];iter!=-1;iter=nx[iter]){
        if(!dfn[v=to[iter]]){
            tarjan(v), low[u] = min(low[u],low[v]);
            if(low[v]>=dfn[u] && (Blo[u]=++block)){
                do tmp[cc++]=vn=sta[now--],Blo[vn]=block,instack[vn]=0;
                while(vn!=v);    //不将 u 出栈，因为一个割点可能属于多个连通分量
                /**
                    在此处 solve
                */

                // 销毁
                cc = 0;
            }
        } else
        if(instack[v]) low[u] = min(low[u], dfn[v]);
    }
}

void solve(int n){
    int i;
    block = 0, mark = 0, now = 0;
    memset(instack,0,sizeof(instack));
    memset(dfn,0,sizeof(dfn));
    memset(Blo,0,sizeof(Blo));
    for(i=1;i<=n;i++)
    if(!dfn[i])tarjan(i);

}
```

2-Sat

```
const NN = 105;

vector <int> vv[NN];
void _addedge(int x, int y) { vv[x].push_back(y) ; }

int dfn[NN], low[NN], stk[NN], blo[NN], block, Cnt, Now;
int ins[NN];

void tarjan(int n)  {
    int i, j;
    low[n] = dfn[n] = ++Cnt, ins[stk[++Now] = n] = 1;
    for (i = 0; i < vv[n].size(); i++)
        if (dfn[j = vv[n][i]] == 0)
            tarjan(j), low[n] = min(low[n], low[j]);
```

```
        else if (ins[j]) low[n] = min(low[n], low[j]);
    if (dfn[n] == low[n] && ++block)
        do blo[i = stk[Now--]] = block, ins[i] = 0;
        while (i != n);
}

vector <int> vvv[NN];
int mth[NN], ind[NN];
int vis[NN];

void Top_sort(int n) {
    memset(vs, 0xff, sizeof(vs));
    for (int i = 1; i <= n; i++) mth[blo[i]] = blo[i + n], mth[blo[i + n]] = blo[i];

    for (int i = 1; i <= n + n; i++)
        for (int j = 0; j < vv[i].size(); j++)
            if (blo[i] != blo[vv[i][j]])
                vvv[blo[vv[i][j]]].push_back(blo[i]), ind[blo[i]]++;

    queue <int> q;
    for (int i = 1; i <= block; i++) if (ind[i] == 0) q.push(i);
    while (q.empty() == 0) {
        int i = q.front();
        q.pop();
        if (vis[i] == -1) vs[i] = 1, vis[mth[i]] = 0;
        for (int j = 0; j < vvv[i].size(); j++) if (--ind[vvv[i][j]] == 0) q.push(vvv[i][j]);
    }
}
```

Nim_SG

```
struct NIM {
    static const int M = 105 ;
    int vis[M], sg[M];
    int _SG(int n) {
        memset(vis, 0, sizeof(vis));
        for (int i = 0; i < n; i++)
            for (int j = 0; j <= i; j++)
                vis[sg[i] ^ sg[j]] = 1;
        for (int i = 0; i < M; i++) if (vis[i] == 0) return i;
    }
    void pre_sg(int n) {
        sg[0] = 0;
        for (int i = 1; i <= n; i++) sg[i] = _SG(i);
    }
    int find_period() {
        for (int res = 1, j; res * 2 < M; res++) {
            for (j = M / 2; j < M; j++) if (sg[j] != sg[j-res]) break;
            if (j == M) return res;
        }
        return -1;
    }
}   nim;
```

Nim_Mul

```
typedef long long LL;
struct NIM_MUL {
    static const int M = 105 ;
    LL pow2[10], sg[M][M];
    int vis[M * M];
    LL _SG(int x, int y) {
        memset(vis, 0, sizeof(vis));
        for (int i = 1; i < x; i++)
            for (int j = 1; j < y; j++)
                vis[sg[i][y] ^ sg[x][j] ^ sg[i][j]] = 1;
        for (int i = 1; i < x; i++) vis[sg[i][y]] = 1;
        for (int j = 1; j < y; j++) vis[sg[x][j]] = 1;
        vis[sg[0][0]] = 1;
        for (int i = 0; i < M * M; i++) if (vis[i] == 0) return i;
    }
    void pre_sg() {
        pow2[0] = 2;
        for (int i = 1; i <= 6; i++) pow2[i] = pow2[i - 1] * pow2[i - 1];
        for (int i = 1; i <= 16; i++)
            for (int j = 1; j <= 16; j++)
                sg[i][j] = _SG(i, j);
    }
    LL nim_mul_pow(LL x, LL y) {
        if (x <= 16) return sg[x][y];
        LL m;
        for (int i = 0; x >= pow2[i]; i++) m = pow2[i];
        LL p = x / m, s = y / m, t = y % m;
        LL d1 = nim_mul_pow(p, s);
        LL d2 = nim_mul_pow(p, t);
        return (m * (d1 ^ d2)) ^ nim_mul_pow(m / 2, d1);
    }
    LL nim_mul(LL x, LL y) {
        if (x < y) return nim_mul(y, x);
        if (x <= 16) return sg[x][y];
        LL m;
        for (int i = 0; x >= pow2[i]; i++) m = pow2[i];
        LL p = x / m, q = x % m, s = y / m, t = y % m;
        LL c1 = nim_mul(p, s);
        LL c2 = nim_mul(p, t) ^ nim_mul(q, s);
        LL c3 = nim_mul(q, t);
        return ((c1 ^ c2) * m) ^ c3 ^ nim_mul_pow(m / 2, c1);
    }
}    nim2;
```

Nim_Graph

```
struct GRAGH_NIM {
    static const int M = 105 ;
    vector <int> vec[M];
    int sg[M], vis[M];
    void clear() {
        for (int i = 0; i < M; i++) vec[i].clear();
        memset(vis, 0, sizeof(vis));
        memset(sg, 0, sizeof(sg));
    }
    void _addedge(int i, int j) { vec[i].push_back(j); vec[j].push_back(i); }
    void dfs(int n) {
        vis[n] = 1;
        for (int i = 0; i < vec[n].size(); i++)
            if (vis[vec[n][i]] == 0) {
                dfs(vec[n][i]);
                sg[n] ^= (1 + sg[vec[n][i]]);
            }
    }
}    gnim;

const int NN = 105;

struct EDGE {
    int i, v;
    EDGE *next,*ani;
    EDGE() {}
    EDGE(int i, int v, EDGE *next, EDGE *ani) : i(i), v(v), next(next), ani(ani) {}
}    *Edge[NN], E[NN << 4];

void _addedge(int i, int j, EDGE &e1, EDGE &e2) {
    e1 = EDGE(j, 1, Edge[i], &e2); Edge[i] = &e1;
    e2 = EDGE(i, 1, Edge[j], &e1); Edge[j] = &e2;
}

int dfn[NN], low[NN], stk[NN], blo[NN], Block, Cnt, Now;
```

```
int ins[NN];

void tarjan(int n) {
    int i;
    low[n] = dfn[n] = ++Cnt, ins[stk[++Now] = n] = 1;
    for (EDGE *p = Edge[n]; p; p = p->next)
        if (p->v) {
            p->ani->v = 0;
            if (dfn[i = p->i] == 0) tarjan(i), low[n] = min(low[n], low[i]);
            else if (ins[i]) low[n] = min(low[n], low[i]);
        }
    if (dfn[n] == low[n] && ++Block)
        do blo[i = stk[Now--]] = Block, ins[i] = 0;
        while (i != n);
}

int CNTE[INF];

void make_gragh(int n) {
    memset(CNTE, 0, sizeof(CNTE));
    for (int i = 1; i <= n; i++)
        for (EDGE *p = Edge[i]; p; p = p->next) {
            if (Blo[i] < Blo[p->i]) gnim._addedge(Blo[i], Blo[p->i]);
            if (Blo[i] == Blo[p->i] && p->vis) CNTE[Blo[i]]++;
        }
    for (int i = 1, m = Block; i <= m; i++) if (CNTE[i] > 1 && CNTE[i] & 1) gnim._addedge(i, ++Block);
}
```

Euler_primes_phi_mu

```
static const int M = 1000005 ;
int prime[M / 12], phi[M], mu[M];
bool primes[M];
void Euler_primes_phi_mu() {
    phi[1] = mu[1] = 1;
    for (int i = 2; i < M; i++) {
        if (primes[i] == 0)
            phi[i] = i - 1, mu[i] = -1, prime[++prime[0]] = i;
        for (int j = 1; j <= prime[0] && prime[j] * i < M; j++) {
            primes[prime[j] * i] = 1;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j], mu[i * prime[j]] = 0;
                break;
              }
            phi[i * prime[j]] = phi[i] * (prime[j] - 1), mu[i * prime[j]] = -mu[i];
        }
    }
}
```

Pollard_Rho

```
typedef long long LL;
map <LL, int> Map;
map <LL, int>::iterator it;

LL GCD(LL a, LL b) { for (LL t; b; t = a % b, a = b, b = t) ; return a ; }
LL mul_mod(LL A, LL B, LL n) {
    LL ans = 0;
    for ( ; B; B >>= 1, A = (A << 1) % n) if (B & 1) ans = (ans + A) % n;
    return ans;
}
LL mod_exp(LL A, LL B, LL n) {
    LL ans = 1LL;
    for ( ; B; B >>= 1, A = mul_mod(A, A, n)) if (B & 1) ans = mul_mod(ans, A, n);
    return ans;
}
bool witness(LL n) {
    LL m = n - 1, a = rand() % m + 1;
    while (m % 2 == 0) m >>= 1;
    if (a = mod_exp(a, m, n), a == 1) return true;
    while (m != n - 1 && a != n - 1)
        a = mul_mod(a, a, n), m <<= 1;
    return a == n - 1;
}
bool miller_rabin(LL n) {
    if (n % 2 == 0) return n == 2;
    for (int i = 0; i < 10; i++)
        if (witness(n) == false) return false;
    return true;
}
LL pollard_rho(LL c, LL n) {
    LL x = rand() % n, y = x, i = 1, k = 2, d;
    do {
        if (i++, d = GCD(n + y - x, n), d > 1 && d < n) return d;
        if (i == k) y = x, k <<= 1;
        x = (mul_mod(x, x, n) - c + n) % n;
    }   while (y != x);
    return n;
}
void rho(LL n) {
    if (n <= 1) return ;
    if (miller_rabin(n)) { Map[n] = 1; return ; }
    LL t;
    do t = pollard_rho(rand() % (n - 1) + 1, n);
    while (t == 1 || t == n);
    rho(t), rho(n / t);
}
void prime_factor(LL n) {
    Map.clear(), rho(n);
    for (it = Map.begin(); it != Map.end(); it++) {
        for (it->second = 0; n % it->first == 0; n /= it->first)
            it->second++;
    }
}
```

Polya

```
typedef long long LL;
//make to prime and phi
static const int M = 1000005 ;
int prime[M / 12], phi[M], mu[M];
bool primes[M];
void Euler_primes_phi() {...}

//prime to make phi
int Phi(int n)
{
    int res = n;
    for (int i = 1; prime[i] * prime[i] <= n; i++)
        if (n % prime[i] == 0)
            for (res -= res/prime[i]; n % prime[i] == 0; n /= prime[i]) ;
    if (n > 1) res -= res / n;
    return res % MOD;
}
//to make phi
int Phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0)
            for (res -= res / i; n % i == 0; n /= i) ;
    if (n > 1) res -= res / n;
    return res % MOD;
}
//
int pow_mod(int a, int n) {
    int ans = 1;
    for ( ; n; n >>= 1, a = (a * a) % MOD) if (n & 1) ans = (ans * a) % MOD;
```

```
    return ans;
}
//use factor to make polay
int cnt[100], fac[100];
void prime_factor(LL n) {
    for (int i = 1; prime[i] * prime[i] <= n; i++)
        if (n % prime[i] == 0) {
            cnt[++cnt[0]] = 0, fac[cnt[0]] = prime[i];
            while (n % prime[i] == 0) cnt[cnt[0]]++, n /= prime[i];
        }
    if (n > 1) cnt[++cnt[0]] = 1, fac[cnt[0]] = n;
}
LL ANS;
void dfs(int step, int fact, int n, int base) {
    if (step == cnt[0]) { ANS = (ANS + pow[base * fact] * phi[n / fact]) % MOD; return ; }
    dfs(step + 1, fact, n, base);
    for (int i = 1; i <= cnt[step + 1]; i++)
        dfs(step + 1, fact *= fac[step + 1], n, base);
}
//to make polay
int polya(int n, int base) {
    int ans = 0;
    for (int i = 1; i * i <= n; i++)
        if (n % i == 0) {
            ans = (ans + pow_mod(base, i) * phi[n / i]) % MOD;
            if (i * i != n)
                ans = (ans + pow_mod(base, n / i) * phi[i]) % MOD;
        }

    ans = (ans * pow_mod(n, MOD - 2)) % MOD;
    return ans;
}
```

FFT

```
const int M = 70005;

const long double PI = acos(-1);
typedef long long LL;
//typedef complex <double> CPX ;

struct CPX {
    double x, y;
    CPX (double x = 0, double y = 0) : x(x), y(y) {}
    CPX operator + (const CPX &o) const { return CPX(x + o.x, y + o.y); }
    CPX operator - (const CPX &o) const { return CPX(x - o.x, y - o.y); }
    CPX operator * (const CPX &o) const { return CPX(x * o.x - y * o.y, x * o.y + y * o.x); }
}   ;

void _FFT(vector <CPX> &A, int op) //慢一点，但是误差更小 {
    int n = A.size();
    for(int i = 0, j = 0, k; i < n; i++) {
        if (j > i) swap(A[i], A[j]);
        for (k = n; j & (k >>= 1); j &= (~k)) ;
        j |= k;
    }
    double pi = PI * op;
    for(int m = 1; m < n; m <<= 1)
        for(int i = 0; i < m; i++) {
            CPX tmp(cos(pi / m * i), sin(pi / m * i));
            for(int j = i; j < n; j += (m << 1)) {
                CPX t = tmp * A[j + m];
                A[j + m] = A[j] - t, A[j] = A[j] + t;
            }
        }
    if (op == -1) for(int i = 0; i < n; i++) A[i].x /= n;
}

```

```
void FFT(vector <CPX> &A, int op) {
    int n = A.size();
    for(int i = 0, j = 0, k; i < n; i++) {
        if (j > i) swap(A[i], A[j]);
        for (k = n; j & (k >>= 1); j &= (~k)) ;
        j |= k;
    }
    double pi = PI * op;
    for(int m = 1; m < n; m <<= 1) {
        CPX tmp(cos(pi / m), sin(pi / m));
        for(int i = 0; i < n; i += (m << 1)) {
            CPX w(1, 0);
            for(int j = i; j < i + m; j++) {
                CPX t = w * A[j + m];
                A[j + m] = A[j] - t, A[j] = A[j] + t;
                w = w * tmp;
            }
        }
    }
    if (op == -1) for(int i = 0; i < n; i++) A[i].x /= n;
}

void CMUL(LL a[], int len1, LL b[], int len2, LL ans[]) {
    int len = 1;
    while (len < (len1 + len2)) len <<= 1;
    vector <CPX> X(len, 0), Z(len, 0);

    for (int i = 0; i < len1; i++) Z[i].x = a[i];
    for (int i = 0; i < len2; i++) X[i].x = b[i];

    FFT(X, 1), FFT(Z, 1);

    for (int i = 0; i < len; i++) Z[i] = Z[i] * X[i];

    FFT(Z, -1);
    for (int i = 0; i < len; i++) ans[i] = Z[i].x + 0.5;
}
```

Det

```
typedef long long LL ;
struct DET {
    static const int M = 205;
    LL a[M][M];
    LL det(int n, int mod) {
        int ans = 1;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++)
            while (a[j][i]) {
                LL t = a[i][i] / a[j][i];
                for (int k = i; k < n; k++) a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                for (int k = i; k < n; k++) swap(a[i][k], a[j][k]);
                ans = -ans;
            }
            if (a[i][i] == 0) return 0;
            ans = ans * a[i][i] % mod;
        }
        return (ans % mod + mod) % mod;
    }
}   ;
```

Simpson

```
struct SIMPSON {
    double f(double x) { return x * x; }
    double simpson(double a, double b) {
        double c = (a + b) / 2;
        return (f(a) + f(c) * 4 + f(b)) * (b - a) / 6;
    }
```

```
    double asr(double a, double b, double ep, double A) {
        double c = (a + b) / 2;
        double L = simpson(a, c), R = simpson(c, b);
        if (fabs(L + R - A) < ep) return L + R;
        return asr(a, c, ep / 2, L) + asr(c, b, ep / 2, R);
    }
    double asr(double a, double b, double ep) {
        return asr(a, b, ep, simpson(a, b));
    }
}    sps;
```

Gauss

```
/*
做完高斯消元后 矩阵的形式为 上三角形
####
 ###
  #
对于浮点型高斯消元
        if (a[i][col] != 0)
        {*****
        }  ->
        if(sgn(a[i][col])!=0){
            double ta = a[i][col]/a[k][col];
            for(j=col; j < var + 1; ++j)
                a[i][j] -= a[k][j]*ta;
        }
*/
int equ, var,a[maxn][maxn],x[maxn]; // 解集.
bool free_x[maxn]; // 判断是否是不确定的变元.
int free_num;

// 高斯消元法解方程组(Gauss-Jordan elimination).(-2 表示有浮点数解，但无整数解，-1 表示无解，0 表示唯一解，大于0 表示无穷解，并返回自由变元的个数)
int Gauss()
{
    int i, j, k,max_r; // 当前这列绝对值最大的行.
    int col; // 当前处理的列.
    int ta, tb,LCM,temp,free_x_num,free_index;
    // 转换为阶梯阵.
    col = 0; // 当前处理的列.
    for (k = 0; k < equ && col < var; k++, col++)
    { // 枚举当前处理的行.
        // 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r = k;
        for (i = k + 1; i < equ; i++)
        {
            if (Abs(a[i][col]) > Abs(a[max_r][col])) max_r = i;
        }
        if (max_r != k)
        { // 与第 k 行交换.
            for (j = k; j < var + 1; j++) swap(a[k][j], a[max_r][j]);
        }
        if (a[k][col] == 0)
        { // 说明该 col 列第 k 行以下全是 0 了，则处理当前行的下一列.
            k--; continue;
        }
        for (i = k + 1; i < equ; i++)
        { // 枚举要删去的行.
            if (a[i][col] != 0)
            {
                LCM = lcm(Abs(a[i][col]), Abs(a[k][col]));
                ta = LCM / Abs(a[i][col]), tb = LCM / Abs(a[k][col]);
                if (a[i][col] * a[k][col] < 0) tb = -tb; // 异号的情况是两个数相加.
                for (j = col; j < var + 1; j++)
                {
                    a[i][j] = a[i][j] * ta - a[k][j] * tb;
                }
            }
        }
    }
    Debug();
    // 1. 无解的情况: 化简的增广阵中存在(0, 0, ..., a)这样的行(a != 0).
    for (i = k; i < equ; i++)
    { // 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换.
        if (a[i][col] != 0) return -1;
    }
    // 2. 无穷解的情况: 在 var * (var + 1)的增广阵中出现(0, 0, ..., 0)这样的行，即说明没有形成严格的上三角阵.
    // 且出现的行数即为自由变元的个数.
    if (k < var)
    {
        // 首先，自由变元有 var - k 个，即不确定的变元至少有 var - k 个.
        for (i = k - 1; i >= 0; i--)
        {
            // 第 i 行一定不会是(0, 0, ..., 0)的情况，因为这样的行是在第 k 行到第 equ 行.
            // 同样，第 i 行一定不会是(0, 0, ..., a), a != 0 的情况，这样的无解的.
            free_x_num = 0; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它们仍然为不确定的变元.
            for (j = 0; j < var; j++)
            {
                if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
            }
            if (free_x_num > 1) continue; // 无法求解出确定的变元.
            // 说明就只有一个不确定的变元 free_index，那么可以求解出该变元，且该变元是确定的.
            temp = a[i][var];
            for (j = 0; j < var; j++)
            {
                if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
            }
            if(temp % a[i][free_index] != 0) return -2;
            x[free_index] = temp / a[i][free_index]; // 求出该变元.
            free_x[free_index] = 0; // 该变元是确定的.
        }
        return var - k; // 自由变元有 var - k 个.
    }
    // 3. 唯一解的情况: 在 var * (var + 1)的增广阵中形成严格的上三角阵.
    // 计算出 Xn-1, Xn-2 ... X0.
    for (i = var - 1; i >= 0; i--)
    {
        temp = a[i][var];
        for (j = i + 1; j < var; j++)
        {
            if (a[i][j] != 0) temp -= a[i][j] * x[j];
        }
        if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解.
        x[i] = temp / a[i][i];
    }
    return 0;
}
```

ExpGcd

```
/*
    拓展欧几里得算法解二元一次不定方程：a*x+b*y=m;
    因为：gcd（a, b）|  a    ,    gcd（a, b）|  b   ;
    所以：gcd（a, b）|  a*x   ,   gcd(a,b) | b*y   ==>  gcd(a,b)|(a*x+b*y)   ==>gcd(a,b)|m ;
    所以要求 a*x+b*y=m，可以先求 a*x+b*y=gcd(a,b).
    对于：a*x+b*y=gcd(a,b)
    1.当 b==0 时，gcd(a,b)=a，此时 x=1,y=0;
    2.先求出 a*x+b*y=gcd(a,b) 的一组解。
    因为    a*x1+b*y1=gcd(a,b)
```

```
            b*x2+a%by2=gcd(b,a%b)
    且       gcd(a,b)=gcd(b,a%b);
    所以有a*x1+b*y1=b*x2+(a-(a/b)*b)*y2
    从而得x1=y2, y1=x2-(a/b)*y2
    然后执行程序段:

*/
void expgcd(int a,int b,int &x,int &y)
{
     if(b==0)
     {
          x=1;
          y=0;
          return ;
     }
     expgcd(b,a%b,x,y);
     int t=x;
     x=y;
     y=t-(a/b)*y;
}



/*
    得出一组解x0, y0;
    又因为此时的解并非是原不定方程a*x+b*y=m 的解并且gcd(a,b)|m
    所以的原不定方程的一组解  x1=x0*(m/gcd(a,b)), y1=y0*(m/gcd(a,b));
    然后又因为原不定方程有无数组解, 并且又有a*(x+(b/gcd(a,b)))+b*(y-(a/gcd(a,b)))=gcd(a,b)
    所以得到原不定方程的所有解为
    x=x1+b/gcd(a,b)*t;
    y=y2-a/gcd(a,b)*t;(t=0,1,2,3,4,5.....................)

*/
```

DanceLinkX

```
const int inf = 0x0f0f0f0f;

struct DLX {
    static const int MC = 350, MR = 1005, M = 3505;
    int D[M], U[M], L[M], R[M], COL[M], ROW[M], S[MC];
    int BEG[MR], END[MR], ANS[MR], N;
    int vis[MC], ans, LIT;

    void init(int n) {
        memset(BEG, 0xff, sizeof(BEG));
        for (int i = 1; i <= n; i++) L[i + 1] = R[i - 1] = D[i] = U[i] = i, S[i] = 0;
        L[1] = R[n] = 0, L[0] = n, N = n + 1;
    }
    void link(int r, int c) {
        D[N] = D[c], U[N] = c, U[D[c]] = N, D[c] = N, COL[N] = c, ROW[N] = r, S[c]++;
        if (BEG[r] == -1) BEG[r] = END[r] = N;
        R[END[r]] = N, L[N] = END[r], R[N] = BEG[r], L[BEG[r]] = N, END[r] = N++;
    }
    void remove_exact(int c) {
        L[R[c]] = L[c], R[L[c]] = R[c];
        for (int i = D[c]; i != c; i = D[i])
            for (int j = R[i]; j != i; j = R[j])
                D[U[j]] = D[j], U[D[j]] = U[j], S[COL[j]]--;
    }
    void resume_exact(int c) {
        L[R[c]] = c, R[L[c]] = c;
        for (int i = U[c]; i != c; i = U[i])
            for (int j = L[i]; j != i; j = L[j])
                D[U[j]] = j, U[D[j]] = j, S[COL[j]]++;
    }

    void remove_repeat(int c) {
        for (int i = D[c]; i != c; i = D[i])
            L[R[i]] = L[i], R[L[i]] = R[i], S[COL[i]]--;
    }
    void resume_repeat(int c) {
        for (int i = U[c]; i != c; i = U[i])
            L[R[i]] = i, R[L[i]] = i, S[COL[i]]++;
    }

    {
    int dfs(int n) {
        int i, now = inf, c;
        if (R[0] == 0) return solve(n), 1;
        for (i = R[0]; i; i = R[i]) if (S[i] < now) now = S[c = i];
        for (remove_exact(c), i = D[c]; i != c; i = D[i]) {
            ANS[n] = i;
            for (int j = R[i]; j != i; j = R[j]) remove_exact(COL[j]);
            if (dfs(n + 1)) return 1;
            for (int j = L[i]; j != i; j = L[j]) resume_exact(COL[j]);
        }
        return resume_exact(c), 0;
    }
    void solve(int n) {
        for (int i = 0; i < n; i++) {
            int j = ROW[ANS[i]];
        }
    }
    }

    {
    int heuristics() {
        memset(vis, 0, sizeof(vis));
        int c, i, j, cnt = 0;
        for (c = R[0]; c; c = R[c])
            if (vis[c] == 0)
                for (cnt++, vis[c] = 1, i = D[c]; i != c; i = D[i])
                    for (j = R[i]; j != i; j = R[j])
                        vis[COL[j]] = 1;
        return cnt;
    }
    int dfs(int n) {
        if (heuristics() + n >= ans) return 0;
        if (R[0] == 0) return ans = n, 1;
        int now = inf, c;
        for (int i = R[0]; i; i = R[i]) if (now > S[i]) now = S[c = i];
        for (int i = D[c]; i !=c ; i = D[i]) {
            remove_repeat(i);
            for (int j = R[i]; j != i; j = R[j]) remove_repeat(j);
            dfs(n + 1);
            for (int j = L[i]; j != i; j = L[j]) resume_repeat(j);
            resume_repeat(i);
        }
        return 0;
    }
    }

    {
    int heuristics() {
        memset(vis, 0, sizeof(vis));
        int c, i, j, cnt=0;
        for (c = R[0]; c <= LIT && c; c = R[c])
            if (vis[c] == 0)
                for (cnt++, vis[c] = 1, i = D[c]; i != c; i = D[i])
                    for (j = R[i]; j != i; j = R[j])
                        vis[COL[j]] = 1;
        return cnt;
    }
```

```
    int dfs(int n) {
        int i, j, now = inf, c;
        if (heuristics() + n >= ans) return 0;
        if (R[0] == 0 || R[0] > LIT) return ans = n, 1;
        for (i = R[0]; i <= LIT && i; i = R[i]) if (S[i] < now) now = S[c = i];
        for (i = D[c]; i != c; i = D[i]) {
            remove_repeat(i);
            ANS[n] = i;
            for (j = R[i]; j != i; j = R[j]) if (COL[j] <= LIT) remove_repeat(j);
            for (j = R[i]; j != i; j = R[j]) if (COL[j] > LIT) remove_exact(COL[j]);
            if (dfs(n + 1)) return 1;
            for (j = L[i]; j != i; j = L[j]) if (COL[j] > LIT) resume_exact(COL[j]);
            for (j = L[i]; j != i; j = L[j]) if (COL[j] <= LIT) resume_repeat(j);
            resume_repeat(i);
        }
        return 0;
    }
    }

}    dlx;
```

Simplex

```
const double eps = 1e-8 ;
int sgn(double x) { return (x < -eps) ? -1 : (x > eps) ; }

struct SIMPLEX {
    static const int ANS_NO = 0, ANS_OK = 1, ANS_INF = 2, M = 805 ;
    double A[M][M], b[M], c[M], OC[M];
    int N[M], B[M], n, m;
    double v, x[M];

    void clear() {
        memset(A, 0, sizeof(A));
        memset(c, 0, sizeof(c));
        memset(b, 0, sizeof(b));
    }

    void pivot(int l, int e) {
        int x, y;
        b[e] = b[l] / A[l][e];
        A[e][l] = 1.0 / A[l][e];
        for (int i = 1; i <= n; i++) if (y = N[i], y != e) A[e][y] = A[l][y] / A[l][e];

        for (int j = 1; j <= m; j++)
            if (x = B[j], x != l) {
                b[x] = b[x] - A[x][e] * b[e];
                for (int i = 1; i <= n; i++) if (y = N[i], y != e) A[x][y] = A[x][y] - A[e][y] * A[x]
[e];
                A[x][l] = -A[x][e] * A[e][l];
            }

        v += b[e] * c[e];
        c[l] = - A[e][l] * c[e];
        for (int i = 1; i <= n; i++) if (y = N[i], y != e) c[y] = c[y] - A[e][y] * c[e];

        for (int i = 1; i <= n; i++) if (N[i] == e) N[i] = l;
        for (int j = 1; j <= m; j++) if (B[j] == l) B[j] = e;
    }

    int opt() {
        for (int l = 0, e = 0, i, j, x, y, tl; 1; ) {
            for (i = 1; i <= n; i++) if (sgn(c[N[i]]) == 1) break;
            if (i > n) return ANS_OK;

            double best = -1e30;
            for (i = 1; i <= n; i++)
                if (y = N[i], sgn(c[y]) == 1) {
                    double now = 1e30, tmp;
                    for (j = 1; j <= m; j++)
                        if (x = B[j], sgn(A[x][y]) == 1) {
                            tmp = b[x] / A[x][y];
                            if (now > tmp || (now == tmp && tl > x)) now = tmp, tl = x;
                        }
                    if ((now * c[N[i]] > best) || (now * c[N[i]] >= best - eps && y < e))
                        best = now * c[N[i]], l = tl, e = y;
                }
            if (best <= -1e29) return ANS_INF;
            pivot(l, e);
        }
    }

    void delete0() {
        int i, j;
        for (j = 1; j <= m; j++) if (B[j] == 0) break;
        if (j <= m) {
            for (i = 1; i <= n; i++) if (sgn(A[0][N[i]])) break;
            pivot(0, N[i]);
        }
        for (i = 1; i <= n && N[i]; i++) ;
        for (n--; i <= n; i++) N[i] = N[i+1];
    }

    int init() {
        int x, y, l = 0;
        for (int i = 1; i <= n; i++) N[i] = i;
        for (int j = 1; j <= m; j++) B[j] = n + j;
        v = 0;

        for (int j = 1; j <= m; j++) if (l == 0 || b[B[j]] < b[l]) l = B[j];
        if (sgn(b[l]) >= 0) return ANS_OK;

        memcpy(OC, c, sizeof(OC));
        memset(c, 0, sizeof(c));
        c[0] = -1; N[++n] = 0;
        for (int j = 1; j <= m; j++) A[B[j]][0] = -1;

        if (pivot(l, 0), opt(), sgn(v) < 0) return ANS_NO;
        delete0();
        memcpy(c, OC, sizeof(c));

        for (int j = 1; j <= m; j++)
            if (x = B[j], sgn(c[x])) {
                v += c[x] * b[x];
                for (int i = 1; i <= n; i++) y = N[i], c[y] -= A[x][y] * c[x];
                c[x] = 0;
            }
        return ANS_OK;
    }

    double simplex() {
        init();
        opt();
        return v;
        for (int j = 1; j <= m; j++) x[B[j]] = b[B[j]];
        //for (int i = 1; i <= n; i++) printf("x[%d] = %.2f\n", i, x[i]);
        //for (int i = 1; i <= n; i++) printf("%.3f%c", x[i], i==n?'\n':' ');
        return ANS_OK;
    }

     /*------------------------------------------------------------------------------------------------
-
        a11 X1 + a12 X2 + a13 X3 + a14 X4 <= b1
        a21 X1 + a22 X2 + a23 X3 + a24 X4 <= b2
        a31 X1 + a32 X2 + a33 X3 + a34 X4 <= b3
```

```
133
134         A[1 + 3][1] = a11; A[1 + 3][2] = a12; A[1 + 3][3] = a13; A[1 + 3][4] = a14; b[1 + 3] = b1;
135         A[2 + 3][1] = a21; A[2 + 3][2] = a22; A[2 + 3][3] = a23; A[2 + 3][4] = a24; b[2 + 3] = b1;
136         A[3 + 3][1] = a31; A[3 + 3][2] = a32; A[3 + 3][3] = a33; A[3 + 3][4] = a34; b[3 + 3] = b1;
137       ----------------------------------------------------------------------------------------------
*/
138
139 }    slx;
```

Hash_Graph

```
09 struct GRAPH_HASH {
10     static const int M = 10005 ;
11     vector <pair <int, int> > edge;
12     int f[M], _f[M], hash[M];
13     int n, A, B, C, D, P, K;
14
15     void clear() { edge.clear(); }
16     void init(int _n, int a, int b, int c, int d, int p, int k) {
18         n = _n, A = a, B = b, C = c, D = d, P = p, K = k;
19     }
20     void _addedge(int i, int j) { edge.push_back(make_pair(i, j)); }
21     void graph_hash() {
23         for (int i = 1; i <= n; i++) {
25             for (int j = 1; j <= n; j++) f[j] = 1;
26             for (int j = 1; j <= K; j++) {
28                 memcpy(_f, f, sizeof(_f));
29                 for (int k = 1; k <= n; k++) f[k] *= A;
30                 for (int k = 0; k < edge.size(); k++) {
32                     f[edge[k].first] += _f[edge[k].second] * B;
33                     f[edge[k].second] += _f[edge[k].first] * C;
34                 }
35                 f[i] += D;
36                 for (int k = 1; k <= n; k++) f[k] %= P;
37             }
38             hash[i] = f[i];
39         }
40         sort(hash + 1, hash + n + 1);
41     }
42 }    g[2];
```

priority_queue

```
01 struct cmp {
02     bool operator() (const NODE &a, const NODE &b) { return a.n > b.n ; }
03 }    ;
05 priority_queue <NODE, vector<NODE>, cmp> small_q;
07 priority_queue <pair <int, int>, vector <pair <int, int> >, greater <pair <int, int> > > small_q;
08 priority_queue <int, vector <int>, greater <int> > small_q;
10 priority_queue <int> large_q;
```

KDT

```
08 typedef long long LL;
09 const LL inf = 0x0f0f0f0f0f0f0f0fLL;
10 const int NN = 100005;
11
12 int idx;
13 struct NODE {
14     static const int KD = 2;
15     int id;
16     _int64 x[KD], dis;
17     void read(int m) { for (int i = 0; i < m; i++) scanf("%I64d", &x[i]); }
18     void show(int m) { for (int i = 0; i < m; i++) printf("%I64d%c", x[i], i == m - 1 ? '\n' : ' '); }
19     bool operator < (const NODE &o) const { return x[idx] < o.x[idx] ; }
20 }    tmp, pt[INF];
21
22 struct KDT {
24     static const int M = 100005 ;
25     int mark[M], spt[M], D;
26     NODE tre[M], ret;
27     void clear() { memset(mark, 0, sizeof(mark)); }
28     template <class TP> TP sqr(TP x) { return x * x; }
29     void build(int l, int r, int cut) {
31         if (l > r) return;
32         int mid = (l + r) / 2;
33         idx = spt[mid] = cut;
34         nth_element(tre + l, tre + mid, tre + r + 1);
35         build(l, mid - 1, (cut + 1) % D);
36         build(mid + 1, r, (cut + 1) % D);
37     }
38     void query(int l, int r, NODE &o) {
40         if (l > r) return;
41         int mid = (l + r) / 2;
42         _int64 dis = 0;
43         for (int i = 0; i < D; i++) dis += sqr(o.x[i] - tre[mid].x[i]);
44
45         if (mark[tre[mid].id] == 0 && dis < o.dis) o.dis = dis, ret = tre[mid];
46
47         _int64 rad = sqr(o.x[spt[mid]] - tre[mid].x[spt[mid]]);
48         if (o.x[spt[mid]] < tre[mid].x[spt[mid]]) {
50             if (query(l, mid - 1, o), rad <= o.dis) query(mid + 1, r, o);
51         } else {
54             if (query(mid + 1, r, o), rad <= o.dis) query(l, mid - 1, o);
55         }
56     }
57 }    kdt;
```

LCT

```
010 typedef long long int64;
011 const int MOD = 51061;
012 const int MAX_N = int(1e5) + 10;
013
014 struct Mark {
015     int64 add, mul; //x*mul+add
016     Mark(int64 add, int64 mul) {
017         this->add = add;
018         this->mul = mul;
019     }
020     Mark() {
021         mul = 1;
022         add = 0;
023     }
024     bool isId() {
025         return mul == 1 && add == 0;
026     }
027 };
028
029 Mark operator*(Mark a, Mark b) {
030     return Mark((a.add * b.mul + b.add) % MOD, a.mul * b.mul % MOD);
031 }
032
033 struct Node {
034     Node*p, *ch[2];
035     bool rev;
036     Mark m;
037     int64 sum, val;
038     int size;
039     bool isRoot;
040     Node*fa;
041     Node() {
042         sum = 0;
043         isRoot = 0;
044         size = 0;
045     }
046     void sc(Node*c, int d) {
```

```
        ch[d] = c;
        c->p = this;
    }
    bool d() {
        return this == p->ch[1];
    }
    void upd() {
        sum = (val + ch[0]->sum + ch[1]->sum) % MOD;
        size = 1 + ch[0]->size + ch[1]->size;
    }
    void apply(Mark a) {
        m = m * a;
        sum = (sum * a.mul + a.add * size) % MOD;
        val = (val * a.mul + a.add) % MOD;
    }
    void revIt() {
        rev ^= 1;
        swap(ch[0], ch[1]);
    }
    void relax();
    void setRoot(Node*f);
} Tnull, *null = &Tnull;

void Node::setRoot(Node*f) {
    fa = f;
    isRoot = true;
    p = null;
}

void Node::relax() {
    if (!m.isId()) {
        REP(i,2)
            if (ch[i] != null)
                ch[i]->apply(m);
        m = Mark();
    }
    if (rev) {
        REP(i,2)
            if (ch[i] != null)
                ch[i]->revIt();
        rev = 0;
    }
}

Node mem[MAX_N], *C = mem;

Node*make(int v) {
    C->sum = C->val = v;
    C->rev = 0;
    C->m = Mark();
    C->ch[0] = C->ch[1] = null;
    C->isRoot = true;
    C->p = null;
    C->fa = null;
    return C++;
}

void rot(Node*t) {
    Node*p = t->p;
    p->relax();
    t->relax();
    bool d = t->d();
    p->p->sc(t, p->d());
    p->sc(t->ch[!d], d);
    t->sc(p, !d);
    p->upd();
    if (p->isRoot) {
        p->isRoot = false;
        t->isRoot = true;
        t->fa = p->fa;
    }
}

void pushTo(Node*t) {
    static Node*stk[MAX_N];
    int top = 0;
    while (t != null) {
        stk[top++] = t;
        t = t->p;
    }
    for (int i = top - 1; i >= 0; --i)
        stk[i]->relax();
}

void splay(Node*u, Node*f = null) {
    pushTo(u);
    while (u->p != f) {
        if (u->p->p == f)
            rot(u);
        else
            u->d() == u->p->d() ? (rot(u->p), rot(u)) : (rot(u), rot(u));
    }
    u->upd();
}

Node*v[MAX_N];
vector<int> E[MAX_N];
int n, nQ;

int que[MAX_N], fa[MAX_N], qh = 0, qt = 0;

void bfs() {
    que[qt++] = 0;
    fa[0] = -1;
    while (qh < qt) {
        int u = que[qh++];
        for (vector<int>::iterator e = E[u].begin(); e != E[u].end(); ++e)
            if (*e != fa[u])
                fa[*e] = u, v[*e]->fa = v[u], que[qt++] = *e;
    }
}

Node* expose(Node*u) {
    Node*v;
    for (v = null; u != null; v = u, u = u->fa) {
        splay(u);
        u->ch[1]->setRoot(u);
        u->sc(v, 1);
        v->fa = u;
    }
    return v;
}

void makeRoot(Node*u) {
    expose(u);
    splay(u);
    u->revIt();
}

void addEdge(Node*u, Node*v) {
    makeRoot(v);
    v->fa = u;
}

```

```
void delEdge(Node*u, Node*v) {
    makeRoot(u);
    expose(v);
    splay(u);
    u->sc(null, 1);
    u->upd();
    v->fa = null;
    v->isRoot = true;
    v->p = null;
}

void markPath(Node*u, Node*v, Mark m) {
    makeRoot(u);
    expose(v);
    splay(v);
    v->apply(m);
}

int queryPath(Node*u, Node*v) {
    makeRoot(u);
    expose(v);
    splay(v);
    return v->sum;
}

int main() {
    scanf("%d%d", &n, &nQ);
    REP(i,n-1) {
        int u, v;
        scanf("%d%d", &u, &v);
        --u, --v;
        E[u].push_back(v);
        E[v].push_back(u);
    }
    REP(i,n)
        v[i] = make(1);
    bfs();
    REP(i,nQ) {
        char cmd;
        scanf(" ");
        scanf("%c", &cmd);
        int i, j;
        scanf("%d%d", &i, &j);
        Node*u = ::v[--i], *v = ::v[--j];
        if (cmd == '+') {
            int c;
            scanf("%d", &c);
            markPath(u, v, Mark(c, 1));
        } else if (cmd == '*') {
            int c;
            scanf("%d", &c);
            markPath(u, v, Mark(0, c));
        } else if (cmd == '/') {
            printf("%d\n", queryPath(u, v));
        } else {
            int k, l;
            scanf("%d%d", &k, &l);
            delEdge(u, v);
            addEdge(::v[--k], ::v[--l]);
        }
    }
}
```

可持久化线段树

```
const int N = 3000000;

int cnt, ls[N], rs[N], sum[N];
int A, B, L[N], R[N], SZ, rt[N];
int n;
void build(int l, int r, int &rt){
    rt = cnt++;
    sum[rt] = 0;
    if(l==r) return ;
    int mid = l+r>>1;
    build(l, mid, ls[rt]);
    build(mid+1, r, rs[rt]);
}

void ins(int last, int &x, int l, int r, int v, int flag){
    x = cnt++, ls[x] = ls[last], rs[x] = rs[last], sum[x] = sum[last]+flag;
    if(l==r) return ;
    int mid = (l+r)>>1;
    if(v<=mid) ins(ls[last], ls[x], l, mid, v, flag);
    else ins(rs[last], rs[x], mid+1, r, v, flag);
}

int query(int l, int r, int k){
    if(l==r) return l;
    int mid = l+r>>1, i,suma=0, sumb=0;
    for(i=0;i<A;++i) suma+=sum[ls[L[i]]];
    for(i=0;i<B;++i) sumb+=sum[ls[R[i]]];
    if(sumb-suma>=k){
        for(i=0;i<A;++i) L[i] = ls[L[i]];
        for(i=0;i<B;++i) R[i] = ls[R[i]];
        return query(l, mid, k);
    }
    else{
        for(i=0;i<A;++i) L[i] = rs[L[i]];
        for(i=0;i<B;++i) R[i] = rs[R[i]];
        return query(mid+1, r, k-(sumb-suma));
    }
}

int ta_query(int l, int r, int k){
    for(A=0;l;l-=l&-l) L[A++] = rt[l];
    for(B=0;r;r-=r&-r) R[B++] = rt[r];
    return query(1, SZ, k);
}

void ta_ins(int x, int v, int flag){
    for(; x<=n; x+=x&-x){
        ins(rt[x], rt[x], 1, SZ, v, flag);
    }
}

vector<int> all;
int getr(int x){
    return lower_bound(all.begin(), all.end(), x) - all.begin() + 1;
}

int a[N], ask[N], lf[N], re[N], K[N];
pair<pair<int,int>, int> q[N];
int main(){
    int Q, i;
    cnt = 0, all.clear();

    scanf("%d%d",&n,&Q);
    for(i=1;i<=n;++i){
        scanf("%d",&a[i]);
        all.pb(a[i]);
    }
    for(i=1;i<=Q;++i){
        char s[5];
        int l,r;
```

```
        scanf("%s%d%d",s,&l,&r);
        q[i] = mp(mp(l,r), -1);
        if(s[0] == 'Q')
            scanf("%d",&q[i].second);
        else all.pb(r);
    }
    sort(all.begin(), all.end());
    all.erase(unique(all.begin(), all.end()), all.end());
    SZ = all.size();
    build(1, SZ, rt[0]);
    for(i=1;i<=n;++i){
        a[i] = getr(a[i]);
        ta_ins(i, a[i], 1);
    }

    for(i=1;i<=Q;++i){
        int l = q[i].first.first, r = q[i].first.second, k = q[i].second;
        if(k!=-1){
            printf("%d\n", all[ta_query(l-1, r, k)-1]);
        }
        else{
            ta_ins(l, a[l], -1);
            a[l] = getr(r);
            ta_ins(l, a[l], 1);
        }
    }
    return 0;
}
```

Splay

```
//don't forget reset
//when u use setc, don't forget upd()
const int MAX_N = 50000 + 10;
const int oo = ~0U >> 1;
struct Node {
    Node*ch[2], *p;
    int size, val, mx;
    int add;
    bool rev;
    void reset();
    bool d() {
        return this == p->ch[1];
    }
    void setc(Node*c, int d) {
        ch[d] = c;
        c->p = this;
    }
    void addIt(int ad) {
        add += ad;
        mx += ad;
        val += ad;
    }
    void revIt() {
        rev ^= 1;
    }
    void relax();
    void upd() {
        size = ch[0]->size + ch[1]->size + 1;
        mx = max(val, max(ch[0]->mx, ch[1]->mx));
    }
} *null;
Node mem[MAX_N], *C;
int top;
void Node::relax() {
    if (add != 0) {
        for (int i = 0; i < 2; ++i) {
            if (ch[i] != null)
                ch[i]->addIt(add);
        }
        add = 0;
    }
    if (rev) {
        swap(ch[0], ch[1]);
        for (int i = 0; i < 2; ++i) {
            if (ch[i] != null)
                ch[i]->revIt();
        }
        rev = 0;
    }
}

Node*make(int v) {
    Node *t;
    if(top) t = ss[top--];else
        t = C++;
    t->ch[0] = t->ch[1] = null;
    t->size = 1;
    t->val = v;
    t->mx = v;
    t->add = 0;
    t->rev = 0;
    return t;
}

void reset(){
    top = 0;
    null = C = mem;
    null = make(-oo);
    null->size = 0;
}

Node*build(int l, int r) {
    if (l > r)
        return null;
    int m = (l + r) >> 1;
    Node*t = make(0);
    t->setc(build(l, m - 1), 0);
    t->setc(build(m + 1, r), 1);
    t->upd();
    return t;
}

Node*root;

Node*rot(Node*t) {
    Node*p = t->p;
    p->relax();
    t->relax();
    int d = t->d();
    p->p->setc(t, p->d());
    p->setc(t->ch[!d], d);
    t->setc(p, !d);
    p->upd();
    if (p == root)
        root = t;
}

void splay(Node*t, Node*f = null) {
    while (t->p != f) {
        if (t->p->p == f)
            rot(t);
        else
            t->d() == t->p->d() ? (rot(t->p), rot(t)) : (rot(t), rot(t));
    }
```

```
    t->upd();
}

Node* select(int k) {
    for (Node*t = root;;) {
        t->relax();
        int c = t->ch[0]->size;
        if (k == c+1)
            return t;
        if (k > c)
            k -= c + 1, t = t->ch[1];
        else
            t = t->ch[0];
    }
}

Node* get(int l, int r) { //(l,r)
    Node*L = select(l);
    Node*R = select(r);
    splay(L);
    splay(R, L);
    return R;
}

void ins(int val){
    if(root == null){
        root = make(val);
        root->p = null;
        root->setc(make(-INF),0);
        root->setc(make(INF),1);
        root->upd();
        return;
    }
    int rk = getr(val);
    Node *t = get(rk,rk+1);
    t->setc(make(val), 0);
    splay(t->ch[0]);
}

void del(int val){
    int rk = getr(val);
    Node* t = get(rk-1,rk+1);
    ss[++top] = t->ch[0];
    t->setc(null, 0);
    t->upd();
    splay(t);
}
```

树链剖分

```
int q[N], fa[N], dep[N], size[N], son[N], pos[N], top[N];
//int eval[N];
void BFS(int root){
    int l, r, u, iter, v, uu, i;

    q[l=r=1] = root;
    fa[root] = 0;
    dep[root] = 0;
    while(l<=r){
        u = q[l++];
        for(i=ghead[u];i!=-1;i=nx[i])
            if((v=to[i]) != fa[u]){
                fa[v]=u, q[++r]=v, dep[v]=dep[u]+1;
                //eval[v] = val[i]; // 边权
            }
    }

    for(i=r;i;i--){
        uu = 0;  //heavy edge
        u = q[i];
        size[u] = 1;
        for(iter=ghead[u];iter!=-1;iter=nx[iter]){
            if((v=to[iter]) == fa[u]) continue;
            size[u] += size[v];
            if(size[v]>size[uu]) uu = v;
        }
        son[u] = uu;
    }

    int now = 0;
    memset(pos,0,sizeof(pos));
    for(i=1;i<=r;i++)
    if(!pos[q[i]]){
        for(u=q[i];u;u=son[u]){
            s[++now] = u,  pos[u] = now, top[u] = q[i];
        }
    }
}

int lca(int x, int y, int &LCA){
    int nx, ny, ans;
    for (ans=11111111;top[x]!=top[y];x=fa[nx]){
        nx = top[x]; ny = top[y];
        if (dep[nx] < dep[ny]) swap(x,y),swap(nx,ny);
        ans = min(ans,ask(pos[nx],pos[x],1));
    }
    //x,y at same line
    //y is the lca of (x,y)
    if (dep[x] < dep[y]) swap(x,y);
    LCA = y;
    // if query edge -> ask(pos[y]+1,pos[x],1)
    ans = min(ans,ask(pos[y],pos[x],1));
    return ans;
}
```

块状数组

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <cstring>
#include <algorithm>
#include <iostream>
#define pb push_back
using namespace std;

const int N = 300005;
const int B = 600;
vector<int> b[B];
int a[N];
int main(){
    int n, m, u, i, j;
    scanf("%d%d%d",&n,&m,&u);
    for(i=0;i<n;++i) scanf("%d",&a[i]);

    int S, cur;
    S = (int)sqrt(n);
    for(i=0,cur=0;i<n;i+=S,++cur){
        for(j=0;j<S&&i+j<n;++j)
            b[cur].pb(a[i+j]);
        sort(b[cur].begin(), b[cur].end());
    }

    while(m--){
        int l, r, v, p, lf, rt, k;
        scanf("%d%d%d%d",&l,&r,&v,&p);
```

```
        --l, --r, --p;
        k = 0;
        for(i=0,cur=0;i<n;i+=S,++cur){
            lf = max(i, l);
            rt = min(i+(int)b[cur].size()-1, r);
            if(rt-lf+1 == b[cur].size())
                k += lower_bound(b[cur].begin(), b[cur].end(), v) - b[cur].begin();
            else
                for(j=lf; j<=rt; ++j)
                    k += a[j]<v;
        }
        int nowv = (long long)u*k/(r-l+1), pos;
        for(i=0,cur=0;i<n;i+=S,++cur)if(p>=i && p<=i+b[cur].size()-1){
            for(j=0;j<b[cur].size();++j) if(b[cur][j]==a[p]){
                pos = j;
                b[cur][j] = nowv;
                a[p] = nowv;
                while(pos+1<b[cur].size() && nowv > b[cur][pos+1])
                    swap(b[cur][pos], b[cur][pos+1]), ++pos;
                while(pos>=1 && nowv < b[cur][pos-1])
                    swap(b[cur][pos],b[cur][pos-1]), --pos;
                break;
            }
        }
    }

    for(i=0;i<n;++i) printf("%d\n", a[i]);

    return 0;
}
```

笛卡尔树

```
/*
    以下代码是基于后缀数组的height 数组建立的笛卡尔树。
    笛卡尔树保证对于A[1..N],如果 i 是对于数组A[0,N-1]的笛卡尔树C(A)是一个二叉树,
    根节点是A 的最小元素，假设 i 为A 数组中最小元素的位置。
    当 i>0 时，这个笛卡尔树的左子结点是A[0,i-1]构成的笛卡尔树,
    其他情况没有左子结点。右结点类似的用A[i+1,N-1]定义。
    注意对于具有相同元素的数组A，笛卡尔树并不唯一

建立方式:
初始栈为空。然后我们在栈中插入A 的元素。
在第 i 步,A[i]将会紧挨着栈中比A[i]小或者相等的元素插入,
并且所有较大的元素将会被移除。
在插入结束之前栈中A[i]位置前的元素将成为 i 的左儿子,
A[i]将会成为它之后一个较小元素的左儿子。
在每一步中，栈中的第一个元素总是笛卡尔树的根。

*/

void set_cartTree(){
    int i, deep = 0;
    sta[0] = 0;
    for (i = 2; i <= n; i++){
        int last = 0;
        while(deep && height[sta[deep]]>height[i]){
            last = sta[deep];
            deep --;
        }
        if (last) tree[i][0] = last;
        tree[sta[deep]][1] = i;
        sta[++deep] = i;
    }
}
```

Computational Geometry

```
const long double eps = 1e-8 ;
const long double PI = acos(-1) ;
int sgn(double x) { return (x < -eps) ? -1 : (x > eps) ; }

//2D PT
struct PT {
    double x, y;
    PT () {}
    PT (double x, double y) : x(x), y(y) {}
    PT operator + (const PT o) { return PT(x + o.x, y + o.y) ; }
    PT operator - (const PT o) { return PT(x - o.x, y - o.y) ; }
    PT operator * (double s) { return PT(x * s, y * s) ; }
    PT operator / (double s) { return PT(x / s, y / s) ; }
    bool operator < (const PT &o) const { return y < o.y - eps || (y < o.y + eps && x < o.x - eps) ; }
    bool operator == (const PT &o) const { return !sgn(y - o.y) && !sgn(x - o.x) ; }
    bool operator != (const PT &o) const { return sgn(y - o.y) || sgn(x - o.x) ; }
    void rd() { scanf("%lf %lf", &x, &y) ; }
    double ag() { return atan2(y, x) ; }
}    ;

bool cmpx(PT a, PT b) { return a.x < b.x - eps || (a.x < b.x + eps && a.y < b.y - eps); }
bool cmpy(PT a, PT b) { return a.y < b.y - eps || (a.y < b.y + eps && a.x < b.x - eps); }

double cpr(PT a, PT b) { return a.x * b.y - a.y * b.x ; }
double dpr(PT a, PT b) { return a.x * b.x + a.y * b.y ; }

double cpr(PT a, PT b, PT c) { return cpr(b - a, c - a) ; }
double dpr(PT a, PT b, PT c) { return dpr(b - a, c - a) ; }

double vlen(PT a) { return sqrt(dpr(a, a)) ; }
double dist(PT a, PT b) { return vlen(a - b) ; }

struct LE {
    PT a, b, v;
    double k;
    LE () {}
    LE (PT a, PT b) : a(a), b(b) { k = (b - a).ag(); v = PT(a.y - b.y, b.x - a.x); v = v / vlen(v) ; }
     bool operator < (const LE &o) const { return sgn(k - o.k) == 0 ? cpr(a, o.a, b) > eps : sgn(k -
o.k) < 0; }
}    ;

struct CLE {
    PT c;
    double r;
    CLE () {}
    CLE (PT c, double r) : c(c), r(r) {}
    bool operator == (const CLE &o) { return c == o.c && !sgn(r - o.r) ; }
    bool operator != (const CLE &o) { return c != o.c || sgn(r - o.r) ; }
    void rd() { c.rd(); scanf("%lf", &r); }
    PT pt(double k) { return c + PT(cos(k), sin(k)) * r; }
    double ag(PT a) { return (a - c).ag() ; }
    double sector0(double k1, double k2) {
        if (k1 > k2 + eps) return sector0(k1, PI) + sector0(-PI, k2);
        return (k2 - k1) * r * r / 2;
    }
    double sector1(double k1, double k2) { return adjust(k2 - k1) * r * r / 2; }
    double adjust(double kk) {
        if (kk < -PI) kk += PI * 2;
        if (kk > PI) kk -= PI * 2;
        return kk;
    }
}    ;

//判两点ab 与直线cd 相对位置，点在直线上0，同侧1，异侧-1
int plside(PT a, PT b, PT c, PT d) { return sgn(cpr(c, a, d) * cpr(c, b, d)); }
```

```
076
077 //判两线段 ab cd 严格相交
078 bool ints_ex(PT a, PT b, PT c, PT d) { return plside(a, b, c, d) == -1 && plside(c, d, a, b) == -1; }
079
080 //判两线段 ab cd 非严格相交
081 bool ints_in(PT a, PT b, PT c, PT d) {
083     int d1 = plside(a, b, c, d), d2 = plside(c, d, a, b);
084     if (d1 == 1 || d2 == 1) return 0;
085     if (d1 == -1 || d2 == -1) return 1;
086     return dpr(c, a, b) < eps || dpr(d, a, b) < eps || dpr(a, c, d) < eps || dpr(b, c, d) < eps;
087 }
088
089 //求直线 ab 和直线 cd 的交点
090 PT ints(PT a, PT b, PT c, PT d) {
092     double v1 = cpr(a, b, c), v2 = cpr(b, a, d);
093     return (c * v2 + d * v1) / (v2 + v1);
094 }
095
096 //点 p 到直线 ab 上的最近点
097 PT ptoline(PT p, PT a, PT b) {
099     PT t(p.x + a.y - b.y, p.y + b.x - a.x) ;
100     return ints(p, t, a, b);
101 }
102
103 //点 p 到直线 ab 距离
104 double disptoline(PT p, PT a, PT b) { return fabs(cpr(p, a, b)) / dist(a, b); }
105
106 //点 p 到线段 ab 上的最近点
107 PT ptoseg(PT p, PT a, PT b) {
109     PT t = p + PT(a.y - b.y, b.x - a.x) ;
110     if (plside(a, b, p, t) == 1) return dist(p, a) < dist(p, b) ? a : b;
111     return ints(p, t, a, b);
112 }
113
114 //点到线段距离
115 double disptoseg(PT p, PT a, PT b) {
117     PT t = p + PT(a.y - b.y, b.x - a.x) ;
118     if (plside(a, b, p, t) == 1) return min(dist(p, a),  dist(p, b));
119     return disptoline(p, a, b);
120 }
121
122 //极角排序
123 PT curp ;
124 bool cmp_angle(PT a, PT b) {
126     int res = sgn(cpr(curp, a, b));
127     if (res > 0) return 1;
128     if (res < 0) return 0;
129     return dist(a, curp) < dist(b, curp) ;
130 }
131
132 bool cmp_angle2(PT a, PT b) {
134     double k1 = a.ag();
135     double k2 = b.ag();
136     return (sgn(k1 - k2) == 0) ? (vlen(a) < vlen(b)) : (sgn(k1 - k2) < 0);
137 }
138
139 //点 v 绕着点 p 逆时针旋转 angle 并放大 scale 倍
140 PT protate(PT v, PT p, double ag, double scale) {
142     PT t = PT(cos(ag), sin(ag)) * scale;
143     v = v - p;
144     p.x += v.x * t.x - v.y * t.y;
145     p.y += v.x * t.y + v.y * t.x;
146     return p;
147 }
148
149 //三角形外心
150 PT circumcenter(PT a, PT b, PT c) {
152     PT u1 = (a + b) / 2;
153     PT u2 = u1 + PT(a.y - b.y, b.x - a.x);
154     PT v1 = (a + c) / 2;
155     PT v2 = v1 + PT(a.y - c.y, c.x - a.x);
156     return ints(u1, u2, v1, v2);
157 }
158
159 //三角形内心
160 PT incenter(PT a, PT b, PT c) {
162     double m, n;
163     m = (b - a).ag();
164     n = (c - a).ag();
165     PT u = a + PT(cos((m + n) / 2), sin(m + n) / 2);
166     m = (a - b).ag();
167     n = (c - b).ag();
168     PT v = b + PT(cos((m + n) / 2), sin(m + n) / 2);
169     return ints(a, u, b, v);
170 }
171
172 //三角形垂心
173 PT perpencenter(PT a, PT b, PT c) {
175
176     PT u = c + PT(a.y - b.y, b.x - a.x);
177     PT v = b + PT(a.y - c.y, c.x - a.x);
178     return ints(c, u, b, v);
179 }
180
181 //重心
182 //到三角形三顶点距离的平方和最小的点，三角形内到三边距离之积最大的点
183 PT barycenter(PT a, PT b, PT c) { return (a + b + c) / 3; }
184
185 //多边形重心
186 PT barycenter(PT p[], int n) {
188     PT ret(0, 0);
189     double t1 = 0, t2;
190     for (int i = 1; i < n - 1; i++)
191         if (sgn(t2 = cpr(p[i+1], p[0], p[i]))) {
193             ret = ret + (p[0] + p[i] + p[i+1]) / 3 * t2;
194             t1 += t2;
195         }
196     if (sgn(t1)) ret = ret / t1;
197     return ret;
198 }
199
200 //计算圆与圆的交点，保证圆与圆有交点，圆心不重合
201 void ints_circle_circle(PT c1, double r1, PT c2, double r2, PT &p1, PT &p2) {
203     double d = dist(c1, c2);
204     double cosA = (r1 * r1 + d * d - r2 * r2) /  (r1 * d * 2);
205     PT v1 = (c2 - c1) * r1 / d;
206     PT v2 = PT(-v1.y, v1.x) * sqrt(1 - cosA * cosA);
207     PT vv = c1 + v1 * cosA;
208     p1 = vv + v2;
209     p2 = vv - v2;
210 }
211
212 //计算直线与圆的交点，保证直线与圆有交点
213 //计算线段与圆的交点可用这个函数后判点是否在线段上
214 void ints_line_circle(PT c, double r, PT a, PT b, PT &p1, PT &p2) {
216     PT p = c + PT(b.y - a.y, a.x - b.x);
217     p = ints(p, c, a, b);
218     double d = dist(p, c), t = sqrt(r * r - d * d) / dist(a, b);
219     p1 = p + (a - b) * t;
220     p2 = p - (a - b) * t;
221 }
222
```

```
223 //两圆公切线切点对应的角度
224
225 //辅助函数
226 void find_tp(PT p, double c, double &ag1, double &ag2) {
228     double v1, v2;
229     v1 = fabs(c) > eps ? (p * c).ag() : p.ag();
230     v2 = acos(fabs(c) / vlen(p));
231     ag1 = v1 - v2;
232     ag2 = v1 + v2;
233 }
234
235 //外公切线(所求角度 t1 t2 均对两圆均适用)
236 void tangent1(PT c1, double r1, PT c2, double r2, double &t1, double &t2) {
238     find_tp(c2 - c1, r1 - r2, t1, t2);
239 }
240
241 //内公切线(所求角度 t1 t2 均对圆 c1 而言,对圆 c2 则需加(减)PI)
242 void tangent2(PT c1, double r1, PT c2, double r2, double &t1, double &t2) {
244     find_tp(c2 - c1, r1 + r2, t1, t2);
245 }
246
247 //判定凸多边形，顶点按顺时针或逆时针给出，允许相邻边共线(n>=3)
248 int is_convex(PT p[], int n) {
250     int s[3] = {1, 1, 1};
251     for (int i = 0; i < n && (s[0] | s[2]); i++)
252         s[sgn(cpr(p[i], p[(i + 1) % n], p[(i + 2) % n])) + 1] = 0;
253     return s[0] | s[2];
254 }
255
256 //判定凸多边形，顶点按顺时针或逆时针给出，不允许相邻边共线
257 int is_convex_v2(PT p[], int n) {
259     int s[3] = {1, 1, 1};
260     for (int i = 0; i < n && s[1] && (s[0] | s[2]); i++)
261         s[sgn(cpr(p[i], p[(i + 1) % n], p[(i + 2) % n])) + 1] = 0;
262     return s[1] && (s[0] | s[2]);
263 }
264
265 //判点在凸多边形内或多边形边上，顶点按顺时针或逆时针给出
266 int inside_convex(PT q, PT p[], int n) {
268     int s[3] = {1, 1, 1};
269     for (int i = 0; i < n && (s[0] | s[2]); i++)
270         s[sgn(cpr(p[i], p[(i + 1) % n], q)) + 1] = 0;
271     return s[0] | s[2];
272 }
273
274 //判点在凸多边形内，顶点按顺时针或逆时针给出,在多边形边上返回 0
275 int inside_convex_v2(PT q, PT p[], int n) {
277     int s[3] = {1, 1, 1};
278     for (int i = 0; i < n && s[1] && (s[0] | s[2]); i++)
279         s[sgn(cpr(p[i], p[(i + 1) % n], q)) + 1] = 0;
280     return s[1] && (s[0] | s[2]);
281 }
282
283 //求多边形 p[]的凸包，严格
284 void make_ch_jm_ex(PT p[], PT o[], int n, int &top) {
286     top = 0;
287     sort(p, p + n);
288     for (int i = 0, j, k = 1; i < 2 * n - 1; i++) {
290         j = (i < n) ? i : 2 * (n - 1) - i;
291         if (i == n) k = top;
292         while (top > k && cpr(o[top - 2], o[top - 1], p[j]) < eps) top--;
293         o[top++] = p[j];
294     }
295     top -- ;
296 }
297
298 //求多边形 p[]的凸包，严格
299 void make_ch_gs_ex(PT p[], PT o[], int n, int &top) {
301     top = 0;
302     curp = p[0];
303     sort(p + 1, p + n, cmp_angle);
304     for (int i = 0; i < n; i++) {
306         while (top > 1 && cpr(o[top - 2], o[top - 1], p[i]) < eps) top--;
307         o[top++] = p[i];
308     }
309 }
310
311 //求多边形面积，p[]为顶点，逆时针为正
312 double poly_area(PT p[], int n) {
314     double ans = 0;
315     for (int i = 1; i < n - 1; i++) ans += cpr(p[0], p[i], p[i + 1]);
316     return ans / 2;
317 }
318
319 //计算多边形外角和，逆时针为正
320 double angsum(PT p[], int n) {
322     double ans = 0;
323     for (int i = 0; i < n; i++) {
325         PT A = p[i], B = p[(i+1)%n], C = p[(i+2)%n];
326         double tmp = PI - acos(dpr(B, A, C) / dist(A, B) / dist(B, C));
327         if (cpr(A, B, C) < -eps) tmp = -tmp;
328         ans += tmp;
329     }
330     return ans;
331 }
332 //半平面交(切割直线为 ab，p[]为原始多边形，顶点数为 n
333 //tmp[]临时用来存放点集，交后多边形仍放入 p[])
334 //多边形点序为逆时针，切割线左侧为有效区
335 void half_plane_its(PT a, PT b, PT p[], int &n, PT tmp[]) {
337     int tot = 0;
338     for (int i = 0; i < n; i++) {
340         int now = sgn(cpr(a, b, p[i]));
341         int next = sgn(cpr(a, b, p[(i+1)%n]));
342         if (now >= 0) tmp[tot++] = p[i];
343         if (now * next == -1) tmp[tot++] = ints(a, b, p[i], p[(i + 1) % n]);
344     }
345     n = tot;
346     for (int i = 0; i < n; i++) p[i] = tmp[i];
347 }
348
349 bool parallel(LE l1, LE l2) { return sgn(cpr((l1.a - l1.b), (l2.a - l2.b))) == 0; }
350
351 PT lineints(LE l1, LE l2) { return ints(l1.a, l1.b, l2.a, l2.b); }
352
353 //l[]为加入的直线，n 为直线个数，p[]为输出的交点，m 为交点数
354 //q[]临时用来存放直线队列
355 //直线为向量，向量左侧为有效区
356 void hpi(LE l[], int n, PT p[], int &m, LE q[]) {
358     int tot = 0;
359     sort(l, l + n);
360     for (int i = 0; i < n; i++) if (i == 0 || sgn(l[i].k - l[tot-1].k)) l[tot++] = l[i];
361
362     n = tot, m = 0;
363     int hh = 0, tt = -1;
364     for (int i = 0; i < n; i++) {
366         if (hh < tt && (parallel(q[hh], q[hh+1]) || parallel(q[tt], q[tt-1]))) return ;
367         while (hh < tt && cpr(l[i].a, l[i].b, lineints(q[tt], q[tt-1])) < eps) tt--;
368         while (hh < tt && cpr(l[i].a, l[i].b, lineints(q[hh], q[hh+1])) < eps) hh++;
369         q[++tt] = l[i];
370     }
371     while (hh < tt && cpr(q[hh].a, q[hh].b, lineints(q[tt], q[tt-1])) < eps) tt--;
372     while (hh < tt && cpr(q[tt].a, q[tt].b, lineints(q[hh], q[hh+1])) < eps) hh++;
```

```
    q[tt+1] = q[hh];
    for (int i = hh; i <= tt; ++i) p[m++] = lineints(q[i], q[i+1]);
    m = unique(p, p+m) - p;
}

//旋转卡壳 凸包直径
double rotating_calipers(PT p[], int n) {
    double ans = 0;
    int q = 0;
    for (int i = 0; i < n; i++) {
        while (cpr(p[i], p[(i+1)%n], p[q]) < cpr(p[i], p[(i+1)%n], p[(q+1)%n])) q = (q + 1) % n;
        ans = max(ans, max(dist(p[i], p[q]), dist(p[i+1], p[q])));
    }
    return ans * ans;
}

//旋转卡壳 两凸包间的最短距离
double rotating_calipers(PT p[], int n1, PT q[], int n2) {
    int pp = 0, qq = 0;
    for (int i = 0; i < n1; i++) if (p[i] < p[pp]) pp = i;
    for (int i = 0; i < n2; i++) if (q[qq] < q[i]) qq = i;
    double ans =1e20;
    for (int i = 0; i < n1 + n2; i++) {
        if (cpr(p[(pp+1)%n1] - p[pp], q[(qq+1)%n2] - q[qq]) < -eps) {
            ans = min(ans, disptoseg(q[qq], p[pp], p[(pp+1)%n1]));
            pp = (pp + 1) % n1;
        } else {
            ans = min(ans, disptoseg(p[pp], q[qq], q[(qq+1)%n2]));
            qq = (qq + 1) % n2;
        }
    }
    return ans;
}

//求最近点对 要先 cmpx 排序
//double dis(PT a, PT b) { return (a.mark == b.mark) ? 1e20 : vlen(a - b); }

double p2pmindis(PT p[], PT o[], int l, int r) {
    if (r == l) return 1e20;
    if (r == l + 1) return dis(p[l], p[r]);

    int mid = (l + r) >> 1;
    double ans = min(p2pmindis(p, o, l, mid), p2pmindis(p, o, mid + 1, r));

    int cnt = 0;
    for (int i = l; i <= r; i++)
        if (p[i].x > p[mid].x - ans + eps && p[i].x < p[mid].x + ans - eps)
           o[cnt++] = p[i];

    sort(o, o + cnt, cmpy);
    for (int i = 0; i < cnt; i++)
        for (int j = i + 1; j < cnt; j++) {
            if (o[j].y - o[i].y > ans - eps) break;
            ans = min(ans, dis(o[i], o[j]));
        }
    return ans;
}

//最小包围圆，增量算法
CLE mincircle(PT p[], int n) {
    random_shuffle(p, p + n);
    CLE ret(p[0], 0);

    for (int i = 0; i < n; i++)
        if (dist(ret.c, p[i]) > ret.r + eps) {
            ret = CLE(p[i], 0);
            for (int j = 0; j < i; j++)
                if (dist(ret.c, p[j]) > ret.r + eps) {
                    PT t = (p[i] + p[j]) / 2;
                    ret = CLE(t, dist(t, p[j]));
                    for (int k = 0; k < j; k++)
                        if (dist(ret.c, p[k]) > ret.r + eps) {
                            t = circumcenter(p[i], p[j], p[k]);
                            ret = CLE(t, dist(t, p[k]));
                        }
                }
        }
    return ret;
}

//计算圆与圆的相交的面积并
double area_circle_circle(PT c1, double r1, PT c2, double r2) {
    if (r1 < r2) swap(c1, c2);
    if (r1 < r2) swap(r1, r2);

    double d = dist(c1, c2);
    if (d > r1 + r2 - eps) return 0;
    if (d < r1 - r2 + eps) return r2 * r2 * PI;

    double cosA1 = (r1 * r1 + d * d - r2 * r2) / (r1 * d * 2);
    double cosA2 = (r2 * r2 + d * d - r1 * r1) / (r2 * d * 2);
    double th1 = acos(cosA1) * 2;
    double th2 = acos(cosA2) * 2;

    double s1 = th1 / 2 * r1 * r1 -  r1 * r1 * sin(th1) / 2;
    double s2 = th2 / 2 * r2 * r2 -  r2 * r2 * sin(th2) / 2;

    return s1 + s2;
}

//三角形剖分，圆与三角形的交
double delaunay_circle(PT a, PT b, PT o, double r) {
    int d1 = (dist(a, o) > r - eps);
    int d2 = (dist(b, o) > r - eps);
    int dd = (disptoseg(o, a, b) < r - eps);

    CLE oo = CLE(o, r);
    double ka = oo.ag(a);
    double kb = oo.ag(b);

    if (dd) {
        if (d1 == 0 && d2 == 0) return cpr(o, a, b) / 2;

        PT p1, p2;
        ints_line_circle(o, r, a, b, p1, p2);
        double k1 = oo.ag(p1);
        double k2 = oo.ag(p2);

        if (d1 == 1 && d2 == 1) {
            if (dist(p1, a) > dist(p2, a)) swap(p1, p2), swap(k1, k2);
            double ans = oo.sector1(ka, k1) + oo.sector1(k2, kb);
            return ans + cpr(o, p1, p2) / 2;
        }

        if (disptoseg(p1, a, b) > eps) swap(p1, p2), swap(k1, k2);

        if (d1 == 1 && d2 == 0) return oo.sector1(ka, k1) + cpr(o, p1, b) / 2;
        if (d1 == 0 && d2 == 1) return oo.sector1(k1, kb) + cpr(o, a, p1) / 2;
    }
    return oo.sector1(ka, kb);
}

//3D PT
```

```
526 struct PT3 {
527     double x, y, z;
528     PT3 () {}
529     PT3 (double x, double y, double z) : x(x), y(y), z(z) {}
530     PT3 operator + (const PT3 o) { return PT3(x + o.x, y + o.y, z + o.z) ; }
531     PT3 operator - (const PT3 o) { return PT3(x - o.x, y - o.y, z - o.z) ; }
532      PT3 operator * (const PT3 o) { return PT3(y * o.z - z * o.y, z * o.x - x * o.z, x * o.y - y * o.
x) ; } // * sin
533     double operator ^ (PT3 o) { return x * o.x + y * o.y + z * o.z ; } // * cos
534     PT3 operator * (double s) { return PT3(x * s, y * s, z * s) ; }
535     PT3 operator / (double s) { return PT3(x / s, y / s, z / s) ; }
536      bool operator < (const PT3 &o) const { return z < o.z - eps ||  (z < o.z + eps && y < o.y - eps)
|| (z < o.z + eps &&y < o.y + eps && x < o.x - eps) ; }
537     void rd() { scanf("%lf %lf %lf", &x, &y, &z) ; }
538 }    ;
539
540 double dpr(PT3 a, PT3 b) { return a ^ b ; }
541 double dpr(PT3 a, PT3 b, PT3 c) { return dpr(b - a, c - a) ; }
542
543 double vlen(PT3 a) { return sqrt(a ^ a) ; }
544 double dist(PT3 a, PT3 b) { return vlen(a - b) ; }
545
546 double volume(PT3 a, PT3 b, PT3 c, PT3 d) { return (b - a) * (c - a) ^ (d - a) ; }
547
548 struct CH3D {
549     static const int M = 1005;
550     struct fac {
551         int a, b, c, ok;
552         fac () {}
553         fac (int a, int b, int c, int ok) : a(a), b(b), c(c), ok(ok) {}
554     }    ;
555
556     int n;
557     PT3 p[M];
558
559     int cnt;
560     fac f[M << 3];
561
562     int vv[M][M];
563
564     double area(PT3 a, PT3 b, PT3 c) { return vlen((b - a) * (c - a)) ; }
565
566     //正：点在面同向
567     double ptof(PT3 pp, fac g) {
569         PT3 m = p[g.b] - p[g.a], n = p[g.c] - p[g.a], t = pp - p[g.a];
570         return (m * n) ^ t;
571     }
572
573     void deal(int j, int a, int b) {
575         int i = vv[a][b];
576         if (f[i].ok) {
578             if (ptof(p[j], f[i]) > eps) dfs(j, i);
579             else {
581                 vv[j][b] = vv[a][j] = vv[b][a] = cnt;
582                 f[cnt++] = fac(b, a, j, 1);
583             }
584         }
585     }
586
587     void dfs(int j, int i) {
589         f[i].ok = 0;
590         deal(j, f[i].b, f[i].a);
591         deal(j, f[i].c, f[i].b);
592         deal(j, f[i].a, f[i].c);
593     }
594
595     int same(int i, int j) {
597         PT3 &a = p[f[i].a], &b = p[f[i].b], &c = p[f[i].c];
598         return (sgn(volume(a, b, c, p[f[j].a])) || sgn(volume(a, b, c, p[f[j].b])) || sgn(volume(a, b
, c, p[f[j].c])))== 0;
599     }
600     //保证前四个点不公面
601     int check() {
603         if (n < 4) return -1;
604
605         int sb = 1;
606         for (int i = 1; i < n; i++)
607             if (vlen(p[0] - p[i]) > eps) {
609                 swap(p[1], p[i]);
610                 sb = 0;
611                 break;
612             }
613         if (sb) return -2;
614
615         sb = 1;
616         for (int i = 2; i < n; i++)
617             if (area(p[0], p[1], p[i]) > eps) {
619                 swap(p[2], p[i]);
620                 sb = 0;
621                 break;
622             }
623         if (sb) return -3;
624
625         sb = 1;
626         for (int i = 3; i < n; i++)
627             if (sgn(volume(p[0], p[1], p[2], p[i]))) {
629                 swap(p[3], p[i]);
630                 sb = 0;
631                 break;
632             }
633         if (sb) return -4;
634         return 0;
635     }
636     //构建三维凸包
637     int construct() {
639         cnt = 0;
640
641         if (check() < 0) return -1;
642
643         for (int i = 0; i < 4; i++) {
645             fac tt((i + 1) % 4, (i + 2) % 4, (i + 3) % 4, 1);
646             if (ptof(p[i], tt) > 0) swap(tt.b, tt.c);
647             vv[tt.a][tt.b] = vv[tt.b][tt.c] = vv[tt.c][tt.a] = cnt;
648             f[cnt++] = tt;
649         }
650
651         for (int i = 4; i < n; i++)
652             for (int j = 0; j < cnt; j++)
653                 if (f[j].ok && ptof(p[i], f[j]) > eps) {
655                     dfs(i, j);
656                     break;
657                 }
658
659         int m = 0;
660         for (int i = 0; i < cnt; i++) if (f[i].ok) f[m++] = f[i];
661         cnt = m;
662     }
663     //表面积
664     double area() {
666         double ret = 0;
667         for (int i = 0; i < cnt; i++) ret += area(p[f[i].a], p[f[i].b], p[f[i].c]);
668         return ret / 2;
669     }
670     //体积
```

```
    double volume() {
        PT3 oo(0, 0, 0);
        double ret = 0;
        for (int i = 0; i < cnt; i++) ret += volume(oo, p[f[i].a], p[f[i].b], p[f[i].c]);
        return fabs(ret / 6);
    }
    //表面多边形数
    int facetcnt() {
        int ans = 0;
        for (int i = 0; i < cnt; i++) {
            int nb = 1;
            for (int j = 0; j < i && nb; j++) nb ^= same(i, j);
            ans += nb;
        }
        return ans;
    }
}    ;

struct SPH {
    PT3 o;
    double r;
    void rd() { o.rd(); scanf("%lf", &r); }
}    ;

//点 p 到直线 ab 距离
double disptoline(PT3 p, PT3 a, PT3 b) { return vlen((b - p) * (a - p)) / dist(a, b); }

//空间上异面直线 ab 与 cd 的距离
double dislinetoline(PT3 a, PT3 b, PT3 c, PT3 d) {
    PT3 t = (b - a) * (d - c);
    return (vlen(t) < eps) ? disptoline(c, a, b) : fabs((d - b) ^ t) / vlen(t);
}

//直线 ab 与球的交点
void ints_line_sphere(PT3 s, double r, PT3 a, PT3 b, PT3 &p1, PT3 &p2) {
    double d = disptoline(s, a, b);
    double t = sqrt(r * r - d * d) / dist(a, b);
    PT3 v = (s - a) * (b - a) * (b - a);
    PT3 p = sgn(d) == 0 ? s : s + v / vlen(v) * d;
    p1 = p + (a - b) * t;
    p2 = p - (a - b) * t;
}

//Longitude 经度 Latitude 纬度
PT3 sphere(double lo, double la) { return PT3(cos(lo) * cos(la), cos(la) * sin(lo), sin(la)); }

//球上两点 a, b 的距离
double ptoponsphere(PT3 a, PT3 b, double r) { return asin( dist(a, b) / 2) * r ; }
```

Plug_Dp

```
#include <tr1/unordered_map>

using namespace std;
using namespace std::tr1;

typedef long long LL;
unordered_map <int, int> umap;

const int inf = 0x0f0f0f0f;

struct STATE {
    int vs[10];
    int m;
    int cn;

    STATE () {}
    STATE (int m) : m(m) {}
    bool operator == (STATE &o) { return cn == o.cn; }

    int hash() { return cn; }

    void decode(int p[]) {
        int now = cn;
        for (int i = 0; i <= m; i++, now >>= 3) p[i] = now & 7;
    }

    void encode(int p[]) {
        cn = 0;
        for (int i = m; i >= 0; i--) cn = (cn << 3) | p[i];
    }

    int minrep(int p[]) {
        memset(vs, 0, sizeof(vs));
        for (int i = 0; i <= m; i++) vs[p[i]] = 1;
        for (int i = 1; i <= m; i++) if (vs[i] == 0) return i;
    }

    void mintrim(int p[]) {
        int cnt = 0;
        memset(vs, 0, sizeof(vs));
        for (int i = 0; i <= m; i++) if (p[i] > 0 && vs[p[i]] == 0) vs[p[i]] = ++cnt;
        for (int i = 0; i <= m; i++) p[i] = vs[p[i]];
    }

    void merge(int px, int py, int p[]) {
        if (px == 0 || py == 0 || px == py) return;
        for (int i = 0; i <= m; i++) if (p[i] == py) p[i] = px
    }

    void fresh(int x, int y, int p[]) { p[x] = p[y] = minrep(p); }

    int tar(int x, int y, int z) { return z == 1 ? x : y ; }
    void trans(int i, int j, int x, int y, int p[]) {
        int xx = p[i], yy = p[j];
        p[i] = (x == 0) ? 0 : tar(xx, yy, x);
        p[j] = (y == 0) ? 0 : tar(xx, yy, y);
    }
}    ;
```

Java

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.lang.*;

public class Solution{
    public static void main(String args[]){
        Scanner cin = new Scanner(System.in);
        int n = cin.nextInt(), i, j, k;
        BigDecimal f[][][] = new BigDecimal[101][101][101];
        BigDecimal g[][] = new BigDecimal[101][101];
        for(j=2;j<=n;++j){
            f[j][j][1] = BigDecimal.valueOf(1.0);
            for(i=n-j+1;i<=n;++i) f[j][j][1] = f[j][j][1].multiply(BigDecimal.valueOf(i));
            f[j][j][1] = f[j][j][1].divide(BigDecimal.valueOf(j));
            g[j][j] = f[j][j][1];

            for(i=j+1;i<=n;++i){
                g[i][j] = BigDecimal.valueOf(0);
                f[i][j][1] = BigDecimal.valueOf(0);
                BigDecimal pl;
                pl = BigDecimal.valueOf(1.0);
                for(k=n-i+1;k<=n-i+j;++k) pl = pl.multiply(BigDecimal.valueOf(k));
```

```
                pl = pl.divide(BigDecimal.valueOf(j));
                for(k=2;k<j;++k) if(i-j>=k)
                    f[i][j][1] = f[i][j][1].add(g[i-j][k].multiply(pl));
                g[i][j] = g[i][j].add(f[i][j][1]);

                for(k=2;k*j<=i;++k){
                    f[i][j][k] = f[i-j][j][k-1].multiply(pl).divide(BigDecimal.valueOf(k), 50, BigDeci
mal.ROUND_HALF_UP);
                    g[i][j] = g[i][j].add(f[i][j][k]);
                }

            }

        }
        BigDecimal tot, ans;
        tot = BigDecimal.valueOf(0);
        ans = BigDecimal.valueOf(0);
        for(j=1;j<=n;++j) for(k=1;k*j<=n;++k) if(f[n][j][k]!=null) tot = tot.add(f[n][j][k]);
        for(j=2;j<=n;++j) for(k=1;k*j<=n;++k) if(f[n][j][k]!=null){
            ans = ans.add((f[n][j][k].multiply(BigDecimal.valueOf(j*k))).divide(tot,50,BigDecimal.ROUN
D_HALF_UP));
        }
        System.out.println(ans);
    }
}
```

最小割集：

1. 求一些特殊要求的割的时候，即使残余网络某边->流量==0，此边不一定是割，因为边的两端不一定因为边->流量==0 就不连通了。
   例如：当使用源点集 S 集 汇点集 T 集 分别标号法时。
   中间的没有被标号的点之间的边是否连通，与其残余流量==0 无关，需要再判断。
   如果，边的至少一端标号，还需注意是否为倒边（即非原始边，乃是后来产生的记录的倒流）。
2. 无向图求割时，是没有倒边的，所有边都是原始边。

有上下界网络流：

1. 对于原图中 x->y 限流[b~l]，则 dif[x]-=b，dif[y]+=b，建边 x->y 容量 l-b。
   对于所有 dif[x]>0，建边超级源点 S->x，容量 dif[x]。
   对于所有 dif[y]<0，建边超级汇点 y->T，容量-dif[y]。
2. 无源网络判断可行流：直接做一次 S->T 的最大流，即可。
3. 有源网络判断可行流：
   判断 s->t 流量为 c 是否有可行流：建边 t->s，容量 c，做一次 S->T 的最大流，即可。
   最小流：二分 s->t 流量 c，判定即可。
   最大流：建边 t->s，容量 inf，第一次做 S->T 的最大流，判定是否有可行解。第二次做 s->t 的最大流，补充最大方案。
4. 方案：
   某边的最终方案=下界 b+边的实际流量。
   仅最大流方案需要做 s->t 的最大流，用以补充可行流以外的方案。
   （可行流受限于限流下界，无法求得残余网络中的 s->t 的其他流）
5. 费用流限流：
   不适合用有上下界网络流方法，容易限制不了。可以用特殊费用(-inf)强行优先流。
   求最大费用时，使用-cost+INF，不一定适用，如果不确定流经过的边数，就减不了 N*INF。

跳舞链：

1. 精确覆盖为首行中 L-R 操作，其他行 D-U 操作。
2. 重复覆盖全为 L-R 操作。
3. 重复覆盖+精确覆盖：可以用重复覆盖+标记数组完成；也可以用重复覆盖+精确覆盖完成，一定要用正确的顺序。
4. DLX 可以用来求解最大子团，先求补集，然后用 DLX 搜索。

求小于等于 N 的与 N 互质的数的和：

1. ANS=N*phi(N)/2
2. 欧拉函数 phi(n)表示小于 n 的正整数中有多少个数与 n 互质。
3. phi(n)=Σ{d|n} mu(d)*n/d
4. n=Σ{d|n} phi(n)

莫比乌斯函数完整定义的通俗表达：

1. 莫比乌斯函数 mu(n)的定义域是 N
2. mu(1)=1
3. 当 n 存在平方因子时，mu(n)=0
4. 当 n 是素数或奇数个不同素数之积时，mu(n)=-1
5. 当 n 是偶数个不同素数之积时，mu(n)=1
6. (n==1)=Σ{d|n} mu(n)

欧拉定理：

a^phi(n) == 1 (mod n)
多面体的面数(F) + 顶点数(V) - 棱数(E) = f(p)
简单多面体 f(p) = 2，带一个洞的多面体 f(p) = 0。
设 G 是连通的平面图，n，m，r 分别是其顶点数、边数和面数，则 v － e + f = 2
(欧拉公式的推广形式) 对于具有 k (k >= 1) 个连通分支的平面图 G，有 v － e + f = k + 1 。

费马定理：

a^(p-1) == 1 (mod p)，p 是质数。

离散对数定理：

g^(x) == g^(y) (mod n) <==> x == y (mod phi(n))

马的哈密尔顿链：

n*n 棋盘存在哈氏链的充要条件是 n>3

马的哈密尔顿圈：

m*n（m<=n）棋盘不存在哈氏圈的充要条件是： m,n 满足下列条件之一 1. m,n 都是奇数 2.m=1,2 或 43.m=3 且 n=4,6,8

Euler 图：

Euler 回路：G 中经过每条边一次且仅一次的回路
Euler 路径：G 中经过每条边一次且仅一次的路径
无向图存在 Euler 回路定理：当它是连通图+顶点度数为偶数
无向图存在 Euler 路径定理：当它是连通图+除两个顶点度为奇数外，其余为偶数
有向图存在 Euler 回路定理：当它是连通图+顶点入度 == 出度
有向图存在 Euler 路径定理：当它是连通图+除一个顶点的入度和出度的差的绝对值小 1 外，其余相等

给出一张混合图（有有向边，也有无向边），判断是否存在欧拉回路

首先是对图中的无向边随意定一个方向，然后统计每个点的入度（indeg）和出度（outdeg），如果（indeg - outdeg）是奇数的话，一定不存在欧拉回路；
如果所有点的入度和出度之差都是偶数，那么就开始网络流构图：
1，对于有向边，舍弃；对于无向边，就按照最开始指定的方向建权值为 1 的边；
2，对于入度小于出度的点，从源点连一条到它的边，权值为（outdeg - indeg）/ 2；出度小于入度的点，连一条它到汇点的权值为（indeg - outdeg）/2 的边；

Pólya 原理计算全部互异的组合状态的个数：

1. Zk（K 不动置换类）：设 G 是 1…n 的置换群。若 K 是 1…n 中某个元素，G 中使 K 保持不变的置换的全体，记以 Zk，叫做 G 中使 K 保持不动的置换类，简称 K 不动置换类。
2. Ek（等价类）：设 G 是 1…n 的置换群。若 K 是 1…n 中某个元素，K 在 G 作用下的轨迹，记作 Ek。即 K 在 G 的作用下所能变化成的所有元素的集合。
3. |Ek|*|Zk|=|G|
4. D(aj) 表示在置换 aj 下不变的元素的个数
5. Σ{1<=j<=n}|Zj|=Σ{1<=i<=s}|D(ai)|
6. Burnside 引理：
   互异的组合状态的个数 L = Σ{1<=j<=n}|Zj| / |G| = Σ{1<=i<=s}|D(ai)| / |G|
7. Pólya 定理：
   设 G 是 p 个对象的一个置换群，用 m 种颜色涂染 p 个对象，则不同染色方案为：
   L = 1/|G| * Σ{1<=i<=n}(m^(c(gi)))
   其中 G={g1 ,…gs} c(gi)为置换 gi 的循环节数(i=1…s) 例如：g=(1)(2)(3)(4) c(g)=4
   Pólya 定理的时间复杂度为 O(s*p)（这里 s 表示置换个数，p 表示格子数）
8. 根据 Burnside 定理，本质不同的方案数为在每个置换下稳定不动的方案的总合除以总置换数。而要得到在置换下稳定不动的方案，即把置换的每个循环节都染上相同的颜色，所以 D(gi)=mc(gi)

Havel 定理：

1. 给定一个非负整数序列{dn}，若存在一个无向图使得图中各点的度与此序列一一对应，则称此序列可图化。进一步，若图为简单图，则称此序列可简单图化。
2. 可图化的判定：d1+d2+……dn=0(mod 2)。关于具体图的构造，我们可以简单地把奇数度的点配对，剩下的全部搞成自环。
3. 可简单图化的判定(Havel 定理):把序列排成不增序,即 d1>=d2>=……>=dn,则 d 可简单图化当且仅当 d'={d2-1,d3-1,……d(d1+1)-1, d(d1+2)，d(d1+3)，……dn} 可简单图化。

简单的说，把 d 排序后，找出度最大的点（设度为 d1），把它与度次大的 d1 个点之间连边，然后这个点就可以不管了，一直继续这个过程，

直到建出完整的图，或出现负度等明显不合理的情况。

N 阶 Nim 游戏：

1. 有 k 堆石子，各包含 x1,x2…xk 颗石子。双方玩家轮流操作，每次操作选择其中非空的若干堆，至少一堆但不超过 N 堆，在这若干堆中的每堆各取走其中的若干颗石子（1 颗，2 颗…甚至整堆），数目可以不同，取走最后一颗石子的玩家获胜。
2. 结论：当且仅当在每一个不同的二进制位上，x1,x2…xk 中在该位上 1 的个数是 N+1 的倍数时，后手方有必胜策略，否则先手必胜。
3. 注意，N 阶情况下，子游戏的 SG 函数只能用来分析胜负，并求必胜策略，而不能求出“N 阶和游戏”的 SG 函数。也就是说，二进制位下分别加和知识得到一组数，而非一个数。

SJ 定理：

1. 对于任意一个 Anti-SG 游戏，如果我们规定当局面中所有的单一游戏的 SG 值为 0 时，游戏结束，则先手必胜当且仅当：
   （1）游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数大于 1；
   （2）游戏的 SG 函数为 0 且游戏中没有单一游戏的 SG 函数大于 1。
2. 有根树中 SG 的求法： 叶子节点的 SG 值为 0；中间节点的 SG 值为它的所有子节点的(SG 值加 1)后的异或和。
3. 有根有环树中 SG 的求法：
   （1） 对于长度为奇数的环，去掉其中任意一个边之后，剩下的两个链长度同奇偶，抑或之后的 SG 值不可能为奇数，所以它的 SG 值为 1
   （2） 对于长度为偶数的环，去掉其中任意一个边之后，剩下的两个链长度异奇偶，抑或之后的 SG 值不可能为 0，所以它的 SG 值为 0
4. 有根有环无向图 SG 的求法：
   将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一个新边；所有连到原先环上的边全部改为与新点相连。这样的改动不会影响图的 SG 值。

单纯形法：

最大化：(∑{i∈N}Ci*Xi)+v

满足：

Xj=bj-(∑{i∈N}Aji*Xi) <==> (∑{i∈N}Aji*Xi)<=bj

Xi>=0 (i∈N)

Xj>=0 (j∈B)

要求：bj>=0

掷骰子中的期望：

m 面的骰子，直到某一面连续出现 n 次，次数的期望 E=((m 的 n 次方)-1)/(m-1)。

快速傅里叶变换

1. FFT 的精度误差会出现在相乘元素值较大的时候，这时逆变换后误差会出现在最小项以及最大项附近。
2. FFT 的项数上限与 >= 乘数项数+被乘数项数。
3. FFT 使用手写的 CPX 比 complex 快 2~3 倍，complex+vector 更慢。
4. FFT 中变化量 tmp 与 w 的求法不一样，更快-误差稍大（如 1 所说），误差更小-稍慢。

平面图：

平面图的边不会超过 3n-5 条。

区分硬币：

1. 一共有 n 枚硬币，其中有 1 枚重量不同，不知轻重。用天平找出。g(k)=(3^k-1)/2。最少次数即为 k，使得 g(k)>=n。
2. 共有 n 球其中 1 个次品，若已知标准球，找出次品轻重：[log3(2n-2)]
3. 共有 n 球其中 1 个次品，若已知标准球，找出次品：[log3(2n-3)]
4. 共有 n 球其中 1 个次品，若不知标准球，找出次品轻重：[log3(2n+2)]
5. 共有 n 球其中 1 个次品，若已知次品轻重，找出次品：[log3n]

皮克定理：

一个计算点阵中顶点在格点上的多边形面积公式：S=a+b/2-1

其中 a 表示多边形内部的点数，b 表示多边形边界上的点数，s 表示多边形的面积。

求和公式：

∑{1<=k<=n}k^2 = n(n+1)(2n+1)/6

∑{1<=k<=n}k^3 = (n(n+1)/2)^2

∑{1<=k<=n}k^4 = n(n+1)(2n+1)(3*n^2+3*n-1)/30

∑{1<=k<=n}k^5 = n^2(n+1)^2(2*n^2+2*n-1)/12

∑{1<=k<=n}k^6 = n(n+1)(2n+1)(3*n^4+6*n^3-3*n+1)/42

∑{1<=k<=n}k^7 = n^2(n+1)^2(3*n^4+6*n^3-n^2-4*n+2)/24

不尽相异元素的全排：

如果在 n 个元素中，有 n1 个元素彼此相同，又有 n2 个元素彼此相同，...，又有 nm 个元素彼此相同(n1+n2+...+nm=n)，那这 n 个元素的全排列称为不尽相异元素的全排列，其排列种数为：n! / (n1!n2!...nm!)

多组组合：

把 n 个不同的元素分成 m 组，第 i 组有 ni 个不同的元素，即(n1+n2+...+nm=n)

这样分组的种数为：n! / (n1!n2!...nm!)

有重复的组合：

从 n 个不同元素中，每次取出 k 个元素，允许重复，不管其顺序合并成一组，

这种组合称为有重复的组合，其组合种数为：C(k，n+k-1)

错排公式：

f[i] = (n-1)*(f[n-2]+f[n-1])

f[1] = 0，f[2] = 1

限位排列公式：

//对于 i,不能放在 i 和 i%n+1

f[i] = f[i-1]*i+(f[i-2]*i+4)/(i-2) (even)

f[i-1]*i+(f[i-2]*i-4)/(i-2) (odd)

catalan 数

h[n] = h[0]*h[n-1]+h[1]*h[n-2]+...+h[n-1]*h[0]

h[n] = h(n-1)*(4n-2)/(n+1)

h[n] = C(2n,n)/(n+1)

h[n] = C(2n,n)-C(2n,n+1)

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452

Lucas 定理

A、B 是非负整数，p 是质数。AB 写成 p 进制:A = a[n]a[n-1]...a[0]，B = b[n]b[n-1]...b[0]

C(A,B)与 C(a[n],b[n])*...C(a[0],b[0])%p 同余

Lucas(n, m, p) = c(n%p, m%p)*Lucas(n/p, m/p, p)

降幂公式

A^x = A^(x % Phi(C) + Phi(C)) (mod C) x>=Phi(C)